教育利用を目的としたオブジェクト指向言語の研究

筑波大学審査学位論文(博士)

2003

兼宗 進

筑波大学大学院
ビジネス科学研究科 企業科学専攻

目 次

第 1 章　序論 **7**

1.1　研究の目的と概要 7

1.2　新教育課程における情報教育 8

1.2.1　総合的な学習の時間 8

1.2.2　中学校の技術・家庭科 9

1.2.3　高等学校の普通教科「情報」 9

1.2.4　その他の教科 10

1.3　情報教育とプログラミング 10

1.3.1　中学校「技術・家庭科」におけるプログラミングの利用 10

1.3.2　高等学校「情報 B」におけるプログラミングの利用 11

1.3.3　教育用プログラミング言語に望まれる性質 11

1.4　研究の概要と本論文の構成 14

第 2 章　情報教育に適したプログラミング言語 **17**

2.1　評価対象 17

2.2　評価方法 18

2.3　プログラミング言語の評価 21

2.3.1　汎用言語 21

2.3.2　高等教育用の言語 25

2.3.3　教育・入門用言語 28

2.3.4　ビジュアル言語 32

2.3.5　例示プログラミング 36

2.4　情報教育に適したプログラミング言語に関するまとめ 39

第3章 プログラミング言語「ドリトル」の設計 41
3.1 プログラミング言語「ドリトル」の設計方針 41
3.1.1 簡潔な構文 42
3.1.2 日本語との対応 42
3.1.3 インクリメンタルなプログラミング 42
3.1.4 テキストによる記述 43
3.1.5 アルゴリズムと構造化 43
3.1.6 オブジェクト指向 43
3.1.7 プロトタイプ方式 43
3.1.8 ネットワークとの通信 44
3.2 言語の設計 44
3.2.1 メッセージ送信 47
3.2.2 メッセージ送信のカスケード 48
3.2.3 リテラルと変数の参照 49
3.2.4 オブジェクトの複製 49
3.2.5 メソッド定義とブロック 50
3.2.6 条件判断 51
3.2.7 条件による繰り返し 52
3.2.8 タイマーによる繰り返し 52
3.2.9 イベント検知 52
3.2.10 外部機器との通信 53
3.2.11 サーバーとの通信 53
3.2.12 標準オブジェクト 54
3.3 ドリトルの実装 55
3.3.1 内部構成 55
3.3.2 ユーザインターフェース 55
3.3.3 インタプリタによる実行 60
3.4 言語仕様の考察 62
3.4.1 クラスを用いないオブジェクト生成 62
3.4.2 メッセージ送信のカスケード 66

3.4.3 中置記法と変数の参照 67
3.4.4 メソッドの定義 67
3.4.5 日本語によるプログラム 68
3.5 情報教育に適したプログラミング言語のまとめ 69

第 4 章 情報教育とプログラミング 71
4.1 高校での実験授業 71
4.1.1 実験概要 71
4.1.2 授業の進め方 72
4.1.3 授業結果 74
4.2 教員による評価 81
4.2.1 調査概要 81
4.2.2 講習の結果 82
4.3 言語仕様の考察 83
4.3.1 タートルグラフィックスの拡張 83
4.3.2 クラスを用いないオブジェクト生成 83
4.3.3 タイマーと並行プログラミング 84
4.3.4 日本語によるプログラム 85
4.4 情報教育とプログラミングのまとめ 85

第 5 章 学校教育での実践と評価 87
5.1 実験概要 87
5.1.1 調査対象 87
5.1.2 調査・実験方法 87
5.1.3 分析方法 88
5.1.4 授業の進め方 90
5.2 授業結果 91
5.2.1 授業ごとのアンケート 91
5.2.2 2 回の定期試験による評価 94
5.2.3 授業実施後のアンケートによる評価 103
5.2.4 作成したプログラムの分析 103

5.3 考察 106
5.4 情報教育とプログラミングに関するまとめ 107

第 6 章 分散プログラミングと情報教育 109

6.1 情報教育と分散プログラミング 109
6.1.1 情報教育における情報通信 109
6.1.2 分散共有ドリトルに求められる特徴 110
6.1.3 既存言語における分散共有機能の検討 111
6.2 分散共有機能 112
6.2.1 基本的な考え方 112
6.2.2 オブジェクトの登録と複製 115
6.2.3 オブジェクトの共有 116
6.3 実装 117
6.3.1 分散共有ドリトルの構成と動作環境 117
6.3.2 ネットワークへの拡張 119
6.3.3 接続 121
6.3.4 登録 121
6.3.5 複製 123
6.3.6 共有 125
6.3.7 共有実行 127
6.3.8 最適化 128
6.4 授業での利用 132
6.4.1 中学校での実験授業 132
6.4.2 企業での新入社員教育 136
6.5 まとめ 138

第 7 章 結論 141

謝辞 153

第1章　序論

1.1　研究の目的と概要

情報社会の急速な発展に伴い、日常生活の中で計算機やネットワークが果たす役割がますます大きくなってきた。それに伴い、「情報」に関するさまざまな側面を生徒に理解させることが学校教育[1] における重要な課題のひとつとなっている。

文部大臣の諮問機関である教育課程審議会は、1998 年に「教育課程の基準の改善に関する答申」[78] を行い、「小学校、中学校および高等学校を通じ、各教科などの学習においてコンピュータなどの積極的な活用を図ること」という新しい指針を示した。これを受けて 1999 年に学習指導要領の改訂が行われ、小学校と中学校では 2002 年度から、高等学校では 2003 年度から、情報教育が開始された。また、「情報化の進展に対応した初等中等教育における情報教育の推進等に関する調査研究協力者会議」は、「情報活用の実践力」「情報の科学的な理解」「情報社会に参画する態度」を情報教育の目標とする最終答申 [74] を行っている。

情報教育の目的のひとつである「情報の科学的な理解」を学ぶ上で、情報を処理する計算機の仕組みを理解することは必要不可欠である。しかし、計算機は他の家電製品などとは異なり、「ソフトウェアによって初めて意味を持つ機械」であることから、計算機を使用したり、分解して観察するだけでは、その原理を理解することは容易ではない。この問題に対する解決策として、大岩ら [43] は、ソフトウェアの開発を体験することによりソフトウェアの特性を理解するアプローチを提案している。

プログラミングによりソフトウェアの開発を体験することは計算機の働きについて効果的に学ぶ手段のひとつであるが、従来はプログラミングの体験を通して学習を行うための環境が十分ではなかった。学校教育の中で使われてきた BASIC や LOGO は、既に 20 年以上前に設計された言語であり、これらの言語を使用してプログラムを作成しても、日常

[1] 本論文では小学校から高等学校において行われる教育を「学校教育」と呼ぶ。「初中等教育」や「K12 (kindergarten and 12-year-education)」と呼ばれることもある。

使われるソフトウェアの理解につながりにくいという問題がある。一方、C++や Java などの汎用言語を用いた場合には、言語自体の習得の難しさから、特定の言語を学ぶことが学習の目的となってしまい、計算機の理解につながりにくいという問題があった。

本論文では、上述の問題点を解決するために、教育利用に適した言語の要件を検討し、学校教育に適した新しいプログラミング言語「ドリトル」を設計・評価する。

1.2 新教育課程における情報教育

新しい学習指導要領 [75] [76] [77] が、小中学校においては 2002 年度から、高等学校においては 2003 年度から実施されている。情報教育に関しては、「情報化の進展に対応した初等中等教育における情報教育の推進等に関する調査研究協力者会議」の最終答申 [74] を受ける形で学習指導要領の改訂が行われた。情報教育に関連する改訂の要点を以下に示す (答申 [74] から要約)。

- 小、中、高等学校段階を通じてコンピュータ等を積極的に活用する
- 小学校段階では、総合的な学習の時間を中心に情報教育を実施する
- 中学校段階では、技術・家庭科の「情報とコンピュータ」を必修とし、発展的内容は生徒 の興味・関心等に応じて選択的に学習する
- 高等学校段階では、新教科「情報」を設け、「情報Ａ」「情報Ｂ」「情報Ｃ」の３科目から１科目を選択必修とする

情報を扱う教科と、そこで扱われる情報教育の内容を以下に示す。

1.2.1 総合的な学習の時間

総合的な学習の時間 [75] [76] [77] は小学校、中学校、高等学校において実施される新しい教科である。

この教科の教科書は用意されず、扱う内容は各学校にまかされている。ただし、「各学校においては、例えば国際理解、情報、環境、福祉・健康などの横断的・総合的な課題、児童の興味・関心に基づく課題、地域や学校の特色に応じた課題などについて、学校の実

態に応じた学習活動を行うものとする」[76] という基準が示されていることから、情報は、この教科の中核をなす重要な役割を期待されていることがわかる。

1.2.2 中学校の技術・家庭科

技術・家庭科 [77] では、従来の教育課程においても情報基礎の内容が扱われていたが、木材加工、電気、金属加工、機械、栽培などと並ぶ一領域でしかなかった。新しい教育課程では、技術分野の半分が「情報とコンピュータ」を扱うことになり、情報の比率が大幅に増加した。

1.2.3 高等学校の普通教科「情報」

従来の教育課程では、「商業科」「工業科」など一部の専門高校だけで「情報」に関連する専門科目が実施されていたが、新しい教育課程では、普通高校において履修する普通教科「情報」 [75] が新設された。

普通教科「情報」は「情報 A」「情報 B」「情報 C」から構成され、生徒はいずれかを履修する。「情報 A」「情報 B」「情報 C」は情報を取り巻く幅広い分野を、それぞれが次のように重点を置いて扱う内容になっている ([75] から要約)。

- 情報 A では、コンピュータや情報通信ネットワークなどの活用を通して、情報を適切に収集・処理・発信するための基礎的な知識と技能を習得させるとともに、情報を主体的に活用しようとする態度を育てる。
- 情報 B では、コンピュータにおける情報の表し方や処理の仕組み、情報社会を支える情報技術の役割や影響を理解させ、問題解決においてコンピュータを効果的に活用するための科学的な考え方や方法を習得させる。
- 情報 C では、情報のディジタル化や情報通信ネットワークの特性を理解させ、表現やコミュニケーションにおいてコンピュータなどを効果的に活用する能力を養うとともに、情報化の進展が社会に及ぼす影響を理解させ、情報社会に参加する上での望ましい態度を育てる。

1.2.4　その他の教科

新教育課程 [75] [76] [77] においては、小学校、中学校、高等学校において実施される上記以外の各教科においても、「生徒がコンピュータや情報通信ネットワークなどの情報手段に慣れ親しみ、適切に活用する学習活動を充実する」ことが求められている。

中村ら [65] が紹介しているように、たとえば、作文やレポート作成でのワードプロセッサ利用、調べもの学習での WWW (World Wide Web) の利用、発表におけるプレゼンテーションソフトの活用など、今まで情報と直接関係を持たなかった教科教育においても、「教育の情報化」が進められている。

よって、学校教育においては学年や教科を問わず、計算機の原理を理解して使いこなす能力が求められているといえる。

1.3　情報教育とプログラミング

情報教育の教材としてプログラミングを活用するためには、使用するプログラミング言語にどのような性質が求められるだろうか。本節では、プログラミングが扱われる教科として中学校の「技術・家庭科」と高等学校の「情報 B」を取り上げ、学習指導要領 [75] [77] の検討を通して、教育用のプログラミング言語に求められる性質を考察する。

1.3.1　中学校「技術・家庭科」におけるプログラミングの利用

中学校「技術・家庭科」[77] の技術分野である「情報とコンピュータ」では、「プログラムの機能を知り、簡単なプログラムの作成ができること」など、プログラミングを含む内容が扱われている。

また、指導計画の作成と内容の取扱いとして、「実践的・体験的な学習活動を中心とし、仕事の楽しさや完成の喜びを体得させるようにすること」「生徒が自分の生活に結び付けて学習できるよう、問題解決的な学習を充実すること」が求められている。これらは教科全体に対する指針であるが、その中で扱われるプログラミングについても適用される。

実践的・体験的な学習活動を行うためには、生徒が容易に理解することができ、段階的に作業を進めて行ける言語が必要である。また、仕事の楽しさや完成の喜びを体得させるためには、生徒が言語を理解して課題を達成できることが必要であり、それに加えて「作

品を作り上げた」と生徒が感じることのできる実用的な課題を容易に記述できる言語である必要がある。

1.3.2 高等学校「情報 B」におけるプログラミングの利用

高等学校「情報 B」[75] では、指導計画の作成と内容の取扱いとして「各科目の目標及び内容等に即してコンピュータや情報通信ネットワークなどを活用した実習を積極的に取り入れること」とし、「総授業時数の 3 分の 1 以上を実習に配当すること」とするなど、実習が重視されている。これらは教科全体に対する指針であるが、その中で扱われるプログラミングについても適用される。

実習でプログラミングを利用するためには、生徒が容易に理解することができ、段階的に作業を進めて行ける言語が必要である。教科の内容としては、「コンピュータ内部での基本的な処理の仕組み及び簡単なアルゴリズム」を扱うことから、計算機の基本的な実行モデルである逐次実行、分岐、繰り返し、手続呼出しなどを扱える言語が必要になる。また、「情報通信と計測・制御の仕組み」を扱うことから、ネットワークや外部機器との通信を扱える言語が必要である。

1.3.3 教育用プログラミング言語に望まれる性質

中学校と高等学校における以上の学習指導要領の内容を考慮すると、情報教育の中でプログラミングを利用するためには、以下の特徴をもつ教育用のプログラミング言語が有効である。

(1) 理解しやすく、習得の時間が短い言語であること

学校教育の中でプログラミングを扱う際には、プログラミングを通して何らかの学習を行うことを目的としており、特定のプログラミング言語の習得を学習の目的とすることはない。たとえば、アルゴリズムの学習にプログラミングを利用する場合は、アルゴリズムを体験的に学ぶことが学習の目的であり、それを記述するプログラミング言語を学ぶことが目的ではない。

しかし、入門用に適していない言語を用いた場合には、生徒が言語自体の学習に時間を取られてしまい、本来の目的であるプログラミングを通した学習に結び付かな

いという問題があった。また、授業を行う教員が必ずしもプログラミングに習熟しているとは限らないため、深い専門知識やプログラミングの経験を必要とする言語は、扱うことができないという問題があった。高等学校の学習指導要領 [75] では、「ソフトウェアの利用技術やプログラミング言語の習得が目的にならないようにする」と明記されており、わかりやすく習得の時間が短い言語が求められていることがわかる。

また、授業の中でプログラミングを扱うときには、言語自体の習得に費やせる時間は限られているという制約もある。たとえば情報科教育法の授業用教科書 [43] で紹介されている高校「情報 B」の年間指導計画案では、プログラミング関連の実習は 10 時間に満たない。このように、限られた時間内で言語を習得して課題解決までを行うことが求められている。

(2) 動かしながら段階的に学べること

計算機はプログラムの命令を解釈し、繰り返しや条件分岐を行いながら順に実行を進めていく。プログラミングに慣れた学習者は計算機が実行する様子を思い浮かべながらプログラムを書き進めることができるが、プログラミングに慣れていない初心者にとって、プログラムが実行される様子を机上で想像しながら学習を進めることは容易ではない。

中谷ら [64] は、短い例題プログラムを動かして、動作を実感しながら進めるプログラミング学習を提案している。また、中学校「技術・家庭科」[77] では「実践的・体験的な学習活動を中心とし、仕事の楽しさや完成の喜びを体得させるようにすること」と明記されており、プログラミングにおいても体験的に進める学習が期待されていることがわかる。

(3) 基本的な計算機の原理を学べること

計算機はプログラムを実行する装置である。高等学校の学習指導要領 [75] で「コンピュータの仕組み、コンピュータ内部での基本的な処理の仕組み及び簡単なアルゴリズムを理解させる」と求められているように、教育用の言語は CPU (Central Processing Unit) が行っている逐次実行や分岐、繰り返しなどの制御構造を体験することができ、変数の概念を扱え、基本的なアルゴリズムを記述できることが必要である。

(4) 日常使うソフトウェアの原理に結び付くこと

今日では、計算機上で動作するアプリケーションソフトウェアは、GUI (Graphical User Interface) 画面で動作することが多い。

入門用の言語においても、グラフィックスや GUI 部品を操作するプログラムを記述できることで、学習のためのプログラムと普段学習者が使用しているアプリケーションソフトウェアが結び付くことが期待できる。大岩ら [43] は、「ふだん目にするグラフィックスや GUI 部品が自分でも操作できるような言語や例題を用いることが、特に『少しだけ』プログラムに接するような生徒には重要」であると指摘している。中学校の学習指導要領 [77] では、「実践的・体験的な学習活動を中心とし、仕事の楽しさや完成の喜びを体得させるようにすること」と明記されており、プログラミングについても日常接する各種ソフトウェアとの結び付きが得られるなど、完成の喜びを得られるような作品を記述できることが期待されていることがわかる。

(5) ネットワークを体験できること

現代の計算機は、単独で動作する形から、他の計算機と通信しながら動作する形態が一般的になっている。たとえば、パーソナルコンピュータはインターネットに接続して利用することが一般的となり、端末の機能を持つ携帯電話も普及が進んでいる。このような背景から、紅林 [49][50] は、通信と制御を行えることが学校教育用の言語にとって重要な性質であると指摘している。

情報通信ネットワークについて、中学校の学習指導要領 [77] では「コンピュータを利用したネットワークについて扱うこと」としており、高等学校の学習指導要領 [75] では「動作を確認できる学習を取り入れるようにする」としていることから、プログラミングを利用する場合には、ネットワークでの通信を扱える言語であることが望ましい。

これまでに数多くのプログラミング言語が提案されてきたにもかかわらず、このような要件をすべて満たしている言語は筆者が調べた限りでは存在していない (詳細は第 2 章で扱う)。本論文では、教育用途に適した新しいプログラミング言語「ドリトル」を提案し、その実用性を評価する。

1.4　研究の概要と本論文の構成

ドリトルは、今日のソフトウェア技術において主要な位置付けを占めるオブジェクト指向 [24] の考えに基づいた教育用プログラミング言語である。従来、教育現場で使われていたBASICやLOGOなどがフローチャートに代表される「手順」という「計算機の考え方」を中心にプログラムを記述するのに対し、ドリトルは動物やボタンなど実世界の「もの」に相当する「オブジェクト」を単位としてプログラムを記述することにより、比較的人間に近い考え方でプログラムを記述することができる。

ドリトルの構文では、日本語の変数名や命令語を用いることで言語的な敷居を低くし、プログラム中で用いる記号を少なくすることで、習得を容易にする工夫を行った。

これらの工夫により、ドリトルは習得が容易な言語になったと考えている。その結果、学校教育の中で扱うことが可能になり、実際に小学校から高等学校の初中等教育、そして大学における高等教育の中で、授業での利用が開始されている。

教育現場に受け入れられた理由としては、言語学習の容易さが挙げられる。従来の言語を利用した授業では、言語を習得する難しさから「言語そのものの学習」で時間を使ってしまっていた。一方、ドリトルを利用した授業では、言語そのものの学習を短時間で終らせて、プログラミングを活用した応用的な学習まで進めるようになった。その結果、特定の言語を学ぶことが目的とならず、学習した言語を使い、プログラミングやソフトウェアといった、計算機の成り立ちに関わる一般的な概念を学ぶことが可能になった。

また、言語自体の学習が容易になったことで扱える対象が広がり、ネットワークを利用した共同プログラミングなど、従来は学校教育の中で扱うことが難しかった領域まで学習を発展させることが可能である。

以下、第2章では、学校教育で行われる情報教育において、言語に求められる性質を考察する。続いて、現在までに提案されている代表的なプログラミング言語について、それらの言語を学校教育で使用したときの問題点を検討し、教育用のプログラミング言語に望まれる性質を考察する。

第3章では、学校教育用に設計したプログラミング言語「ドリトル」について解説する。最初に言語の設計方針を述べる。続いて言語の構文を説明し、仕様について議論を行った後、実装と内部構造についての設計を解説する。

第4章では、ドリトルを用いて行った少人数の実験授業について述べる。授業は、数人

の高校生と十数人の学校教員を対象に行った。学習者がドリトルを学習する様子の観察と、学習者が作成したプログラムの分析から、設計した言語の仕様について考察を行う。

第5章では、学校で正規の授業として行った実験授業について述べる。この授業は公立中学校で教員が行い、1学年132人の生徒が学習した。理解度を調べるために、2種類のアンケート調査と2回の定期試験による調査を行った。これらの結果と生徒が作成したプログラムの分析から、ドリトルを中学校の授業で使用した場合の学習に関する理解度を評価する。

第6章では、ドリトルを拡張した分散共有ドリトルについて述べる。分散共有ドリトルでは、ネットワークを利用して、複数のドリトルがオブジェクトを交換・共有することができる。この機能を利用して、複数の生徒が協調して動作するプログラムを作成することが可能である。最初に機能の設計を行った後、実装を詳細に説明する。続いて、中学生と社会人を対象に行った実験授業を紹介し、分散共有ドリトルを用いたプログラミング学習を考察する。

第7章では、全体を総括し、結論を記す。

第2章　情報教育に適したプログラミング言語

学校教育の中でプログラミングを利用するためには、適切なプログラミング言語が不可欠となる。本章では、学校教育用で使われる言語に求められる性質を検討した後、既存の代表的なプログラミング言語の評価を行い、教育用に適したプログラミング言語について考察を行う。

2.1　評価対象

企業や大学等を含めた教育環境で、プログラミングの入門用に使われている言語について評価を行う。言語は用途と機能から、次の 5 つのグループに分類して評価した。製品開発に用いられる汎用言語、高等教育用に設計された言語、入門・教育用の言語、図形など視覚的に操作するビジュアル言語、例示により間接的に意図を伝える例示プログラミングである。

- 汎用言語
  企業などで開発用に使われている言語であり、入門用、教育用としての考慮はなされていない。汎用言語の例として、代表的な C++と Java を評価する。

- 高等教育用言語
  大学等の高等教育では汎用言語を利用する場合が多いが、学習の敷居を下げる目的から、いくつかの高等教育用の言語が提案されている。高等教育用言語の例として、論文が公開されている Blue と若葉を評価する。

- 入門・教育用言語
  初中等教育では、汎用言語は難易度が高すぎて使用できないという理由から、入門

用、教育用の言語が利用されて来た。入門・教育用言語の例として、学校教育で利用されることのある BASIC と LOGO を評価する。

- ビジュアル言語
  入門用のアプローチとして、文字ではなく図形の組み合わせによりプログラムを記述する言語が存在する。ビジュアル言語の例として、画面上でプログラミングを行う LEGO MindStorms と、実世界のブロックを用いてプログラミングを行うアルゴブロックを評価する。

- 例示プログラミング
  明示的にプログラムを記述する代りに、望ましい動作の例を示しそれを計算機が学習する形で、間接的に望ましい動作のプログラミングを実現するアプローチが存在する。例示プログラミングの例として、論文の公開されている KidSim と Gamut を評価する。

2.2 評価方法

この節では、教育利用の観点から言語を評価するための基準を示す。評価基準は、教育用プログラミング言語に求められる性質 (第 1.3 節) を満たすために必要な言語的な要件を検討した。

(1) わかりやすく、習得の時間が短いこと (理解しやすさ)
  言語を短い時間で習得するためには、プログラミング言語の概念や構文が簡潔であり、容易に学べることが望ましい。市川 [40] は、わかりやすい言語の性質として、プログラムを簡潔な構文で記述することができ、プログラム内で使われる構文要素 (データ型や変数など) の煩雑な宣言が不要で、高度な概念 (ポインタ、スタック、クラスなど) を理解しなくてもプログラミングを行える言語が望ましいと指摘している。

  - 構文が簡潔であること
    テキストや黒板で説明することを考えると、入門用のサンプルプログラムは数行程度で記述できることが望ましい。各種の宣言や定義が必要になる言語は、学習者が本質的でない部分に注意を奪われてしまう可能性がある。また、記号

を多く使う言語や、“ [”, “ (”, “ {” など、複数の似た記号を使い分ける必要がある言語は、記述のミスによる構文エラーを起こしやすいため、学習意欲を損なわせることにつながりやすい。

- 高度な概念が不要であること
  プログラミングを始める前に、クラスやポインタなどの難しい概念を学ぶ必要がある言語では、初心者に必要以上の壁を感じさせてしまう可能性があり望ましくない。

(2) 動かしながら学べること (対話性)
プログラムの動作を少しずつ確認しながら学習を進めるためには、プログラムを数行から記述することができ、動作の結果を視覚的に確認しながら段階的に拡張できる言語であることが望ましい。市川 [40] は、事前に理解すべき概念の多い言語では多くのことを学習してからでないと動かしてみることができないため、動かしながらプログラミングを学習する入門用の用途に適さないことを指摘している。

- 段階的に試せること
  入門用のサンプルプログラムを数行で記述でき、1 行ずつ増やしながら試していける言語が望ましい。プログラムを試すたびに長い行数を入力する必要があり、1 文字でも間違えたときにエラー個所を探すのが大変な言語は、学習者を混乱させ、興味を失わせる可能性がある。
- 結果を視覚的に確認できること
  プログラムを実行したときに、自分のプログラムがどう動いたのかを色や動きで視覚的にフィードバックできる言語が望ましい。数値や文字だけが出力される言語では、プログラムの動きや誤りを感覚的につかみにくいという問題がある。

(3) 基本的な計算機の原理を学べること (構造化)
計算機の基本的な処理の仕組みと簡単なアルゴリズムを学習する必要から、CPU が行う逐次実行や繰り返しといった基本的な処理と、それらの処理を組み合わせて実用的なプログラムを記述するのに必要な構造化の概念を扱える言語が望ましい。Bork[3] は、構造化プログラミングを行えることは教育用言語の満たすべき性質であると指摘している。

- 基本的なアルゴリズムを記述できること
  CPUが行っている逐次実行や分岐、繰り返しなどの制御構造を体験することができ、変数の概念を扱え、基本的なアルゴリズムを記述できることが必要である。学習者がプログラムを明示的に記述しない言語では、計算機がアルゴリズムを実行する様子を示すのは難しい。

- プログラムを構造化できること
  プログラムが長くなると、理解することが急速に難しくなる。基本的な命令を組み合わせて実用的なソフトウェアを作成するためには、構造化 [7] されたプログラムを記述することが不可欠である。手続きや関数などの構造を作ることにより、いちどに理解する部分が短くなり、プログラム全体を分割して理解できるようになる。構造を作れない言語は、学習者が常にプログラム全体を把握する必要があり望ましくない。

(4) 日常使うソフトウェアの原理に結び付くこと (オブジェクト指向)
今日では、パーソナルコンピュータ上で動作するアプリケーションソフトウェアは、GUI画面で動作することが一般的である。入門用の言語においても、グラフィックスやGUI部品を操作するプログラムを記述できることで、学習のためのプログラムと普段学習者が使用しているソフトウェアが結び付くことが期待できる。また、ゲームやストーリー性のあるアニメーションなど、作品として達成感を感じられるようなひとまとまりの動作をするプログラムを記述するためには、構造化に加えて、オブジェクト指向 [24] の概念を扱えることが有効である。

- 視覚的なオブジェクト部品が用意されていること
  少ない手間でプログラミングを体験するために、GUI部品のような学習に活用できる初心者用のオブジェクト部品があらかじめ用意されていることが望ましい。

- オブジェクト部品を組み合わせてプログラムを作れること
  オブジェクト指向言語では、オブジェクトにメソッドとしてプログラムを定義することができる。あらかじめ用意されたオブジェクトと自作のオブジェクトを部品のように組み合わせることで、高度なプログラムを容易に記述できる言語が望ましい。

(5) ネットワークを体験できること (ネットワーク)

情報通信の仕組みを体験的に学習するために、ネットワークの通信を容易に扱える言語が必要である。また、オブジェクト指向言語では、数値などの基本的なデータに加えて、オブジェクトを対象とした通信ができることが望ましい。

- ネットワークの通信を行えること
  ネットワークの技術的な専門知識や高度なプログラミングの知識を必要とせずに通信を体験できる言語が望ましい。C や Java などの言語で他の計算機と通信するプログラムを記述する場合には、一般に高度な知識が必要となり望ましくない。
- ネットワーク上でオブジェクトを転送できること
  オブジェクト指向言語では、オブジェクトが操作の対象となる。ネットワークで通信を行う場合には、数値などの基本的なデータに加えて、オブジェクトを対象とした通信ができることが望ましい。

次節では、これらの評価基準に沿って言語の評価を行う。

2.3 プログラミング言語の評価

本節では、学校教育で使うことを想定して、実際のプログラミング言語を分析する。対象とした言語は、汎用言語、高等教育用言語、教育・入門用言語、ビジュアル言語、例示プログラミングである。

テキストで記述する言語については、比較のために “Hello World” という文字列を表示する小さなプログラムを例として示した。また、必要に応じてその言語の特徴を表すプログラム例を示す。

2.3.1 汎用言語

汎用言語 [51] は特に教育用途や入門用途を前提としていない言語であり、実社会において製品開発等に使われている。古くは大型計算機上で動作していた計算用言語 (FORTRAN など)、事務処理用言語 (COBOL など)、計算と事務処理に両用できる言語 (PL/I など) が

代表的であった。現在ではオペレーティングシステムやアプリケーションプログラム等のシステム開発に用いられる言語として、C [18] や、オブジェクト指向言語である C++ [35]、Java [13] などが多く使われている。オブジェクト指向言語としては、Smalltalk[12] やその実装である Squeak[16] なども存在する。これらは専門の教育を受けたプログラマが使うことを想定した言語であり、特に初心者のための考慮はなされていない。

以下では、現代の製品開発に広く使われている代表的なオブジェクト指向言語として、Java と C++を取り上げ検討する。

Java

Java [13] は 1995 年にサン・マイクロシステムズ社から発表された言語である。WWW ブラウザ上でアプレットとして実行できるなどインターネットとの親和性が高く、次の特徴を持つ。

- オブジェクト指向言語である
- 移植性が高く、多くのプラットフォームで動作する
- セキュリティが考慮されており安全性が高い
- 複数の処理を同時に行うマルチスレッド処理が可能である

図 2.1 に Java のプログラム例を示す。

```
public class HelloWorld {
  public static void main(String args[ ]) {
    System.out.println("Hello World");
  }
}
```

図 2.1: Java のプログラム例 (Hello World)

このプログラムを入門用のサンプルとして使うことを考えると、次のような問題があることがわかる。

1. 高度な概念を理解する必要がある
   もっとも単純な入門用のプログラムであるにも関わらず、「public (公開スコープ)」「class (クラス)」「static (静的スコープ)」「void (返値なし)」「String (文字列クラス)」「System.out (システムクラスの出力ストリーム)」といった高度なキーワードが出現している。これらの意味をすべて理解できるのはしばらく学習を続けた後である。理解できるまでは学習の過程で「おまじない」として、意味のわからない文字列を記述し続ける必要があり、教育的に望ましくない。

2. 構文が難しい
   記号として「( )」「{ }」「;」「" "」「[ ]」など多くの種類を区別して記述する必要がある。また、英字も大文字と小文字を区別して入力する必要があるなど、初心者の構文エラーを誘発しやすい。

3. 多重の入れ子構造がある
   中括弧「{ }」で区切られる形で、「外側にクラス定義があり、その内部に関数定義、その内部に実行文」という多重の入れ子構造でプログラムが記述されている。このような複数の概念にまたがる多重の入れ子構造は、初心者にとって理解することが困難である。

4. 段階的に試せない
   画面にメッセージを表示する図 2.1 のような最も単純なプログラムでも、複数行の記述が必要になる。1 行ずつ記述を増やしながら動作を確認していく、インクリメンタルなスタイルを取りにくいという問題がある。

Java は当初からインターネットを想定して設計されるなど優れた性質を持っているが、そのまま教育・入門用として使うことは容易ではない。

C++

C++ [35] は、代表的なシステム開発用言語である C を拡張する形で AT&T で開発された。図 2.2 に C++のプログラム例を示す。

このプログラムを入門用のサンプルとして使うことを考えると、次の問題があることがわかる。

```
#include <iostream.h>
int main( )
{
  cout << "Hello World" << endl;
}
```

図 2.2: C++のプログラム例 (Hello World)

1. 高度な概念を理解する必要がある
   もっとも単純な入門用のプログラムであるにも関わらず、「`#include` (ヘッダファイルの宣言)」「`int` (整数型)」「`cout` (標準出力ストリーム)」「`<<` (cout 用の演算子)」「`endl` (ファイル末尾を示す処理子)」といった高度なキーワードが出現している。これらの意味をすべて理解できるのはしばらく学習を続けた後である。理解できるまでは学習の過程で「おまじない」として、意味のわからない文字列を記述し続ける必要があり、教育的に望ましくない。

2. 構文が難しい
   記号として「`#`」「`< >`」「`( )`」「`{ }`」「`<<`」「`;`」「`" "`」など多くの種類を区別して記述する必要がある。また、英字も大文字と小文字を区別して入力する必要があるなど、初心者が構文エラーを誘発しやすい。

3. 入れ子構造がある
   中括弧「`{ }`」で区切られる形で、「関数定義の内部に実行文」という入れ子構造でプログラムが記述されている。このような入れ子構造は、初心者にとって理解することが容易ではない。

4. 段階的に試せない
   画面にメッセージを表示する図 2.2 のような最も単純なプログラムでも、複数行の記述が必要になる。1 行ずつ記述を増やしながら動作を確認していく、インクリメンタルなスタイルを取りにくいという問題がある。

Kölling et al. [19] と Hitz et al. [15] は、大学において入門用言語として用いた経験から、C++を次のように評価している。

- 構文が難しく、大学の入門コースで採用したところ、40%の学生が早期に脱落した。
- 既存の言語であるCを拡張したため、仕様に一貫性がない。
- オブジェクトを生成する方法が「自動変数」「生成命令」「代入」の3通りもあり、わかりづらい。
- 明示的なメモリ管理がある。不必要に低い (抽象的な階層の低い) レベルを扱う必要があり、理解が難しい。
- クラス階層や仮想関数という高度な概念の理解が必要であり、大学生でも理解が難しい。
- プログラムの行数が多くなり、可読性が悪い。

C++は現在、商用のアプリケーション開発にもっとも使われている言語のひとつである。しかし、習得の難易度が高く大学などの高等教育においても学ぶことが容易ではないことから、学校教育の中で教育・入門用として使うには適切な言語ではない。

2.3.2 高等教育用の言語

大学などの高等教育においては、CやJavaなど汎用の言語がそのまま利用されることが多いが、習得を容易にすることを目的とした専用の言語も提案されている。以下では、高等教育用言語として論文が公開されているBlue [20]と若葉 [48]を取り上げ検討する。

Blue

Blue [19] [20]は、大学等の高等教育用に開発された言語である。

図 2.3にBlueのクラス定義例を示す。このプログラムを入門用のサンプルとして使うことを考えると、次の問題があることがわかる。

1. 高度な概念を理解する必要がある
   もっとも単純な入門用のプログラムであるにも関わらず、「`class` (クラス)」「`internal` (内部宣言)」「`var` (変数宣言)」「`String` (文字列クラス)」「`interface` (公開宣言)」といった高度なキーワードが出現している。これらの意味をすべて理解できるのは

```
class HelloWorld is
  internal
  var
    _str :  String
  interface
    creation () is
    do
      _str := "Hello World"
    end creation
end class
```

図 2.3: Blue のプログラム例 (Hello World)

しばらく学習を続けた後である。理解できるまでは学習の過程で「おまじない」として、意味のわからない文字列を記述し続ける必要があり、教育的に望ましくない。

2. 入れ子構造がある
   「外側にクラス定義があり、その内部に関数定義、その内部に実行文」という多重の入れ子構造でプログラムが記述されている。このような複数の概念にまたがる多重の入れ子構造は、初心者にとって理解することが困難である。

3. 段階的に試せない
   画面にメッセージを表示する図 2.3 のような最も単純なプログラムでも、複数行の記述が必要になる。1 行ずつ記述を増やしながら動作を確認していく、インクリメンタルなスタイルを取りにくいという問題がある。

Blue は高等教育用に設計された言語だが、学校教育用の言語として見たときに、特に優れた点は存在しない。

若葉

若葉 [48] は、大学における高等教育用に設計された言語である。図 2.4、図 2.5 に若葉のプログラム例を示す。

```
print "Hello World";
```

図 2.4: 若葉のプログラム例 (Hello World)

```
func sum(num max){
  num x; x = 0;
  loop {
    x = x + 1;
    if (x > max)
      exit x;
  };
}
```

図 2.5: 若葉のプログラム例 (数の計算)

これらのプログラムを入門用のサンプルとして使うことを考えると、次の問題があることがわかる。

1. 初心者の誤りを考慮していない
   繰り返しが loop 構文を用いた無限ループしか用意されていないため、図 2.5 で if 文による脱出条件を書き忘れたり、脱出条件に誤りがあると、実行が無限ループに陥ってしまい、学習者の混乱を招く可能性がある。

2. 言語仕様に初心者を混乱させる要素がある

   - 真偽値を数値で表す言語仕様となっているが、真偽値を数値で表すことは、プログラミングの誤りを誘発するため、初心者にとって望ましくない。
   - 制御構文が整備されていないため、他の言語で簡単に書ける定数回の繰り返しや条件付き繰り返しを図 2.5 のように無限ループで記述しなければならず、学習者への負担が大きい。
   - 関数への引数がすべて参照渡しになっている。図 2.5 において、関数 sum の中で引数 max の値を修正すると、この関数を呼び出した側の max の値も修正さ

れてしまう。このように引数が参照渡しであるため、情報隠蔽が完全ではない。そのため、呼び出した関数の副作用により、呼び出し側の変数の値が変更されてしまう危険が存在する。

3. 現代的な概念が含まれていない
   基本構造は古典的な手続き型言語のままであり、現代のプログラミングを学ぶ上で重要になるオブジェクトやカプセル化の概念が用意されていない。

若葉はシンプルな構文を採用しており、アルゴリズムの学習などで利用することは可能だが、オブジェクト指向の概念やグラフィックスや GUI 部品を活用する機能が考慮されていないため、学校教育においては用途が限られてしまうという問題がある。

2.3.3 教育・入門用言語

学校教育においては、BASIC [17] や LOGO [26] などの言語が広く用いられてきた。これらの言語は 1960 年代に設計が行われており、基本的な機能が理解しやすいという特徴を持つが、近年の新しい概念は導入されていないという問題がある。萩谷 [67] は、「BASIC のような貧弱な言語が、こうも長く生き残っているというのは驚嘆に値する」と指摘している。

その他、日本語でプログラムを記述する言語として、Mind[46]、ひまわり [80]、TTS[70] などが存在する。これらは日本語の文字や記号を使ってプログラムを記述できるが、学校教育ではほとんど使われていない。言語の問題点としては、BASIC や LOGO の評価が当てはまる。

以下では、学校教育で使われることのある教育・入門用言語として、BASIC と LOGO を取り上げ検討する。

BASIC

BASIC [17] は、Kemeny と Kurts が 1964 年に考案した言語である。“Beginner’s All-purpose Symbolic Instruction Code” という名前が示すように、当初から初心者の入門用言語を意識して設計された。BASIC は初期の一部のパーソナルコンピュータに組み込まれた形で提供され、インストールなどを行うことなくインタプリタとして 1 行ずつの対

話的な実行が可能であったため、入門用の言語として広く利用されてきた。一方で、長谷川 [69] は、BASIC には標準的な言語仕様が存在しなかったため、数多くの同名異体の BASIC が存在することを指摘している。

基本的な BASIC では、プログラムの構造は行番号を単位としたジャンプ命令が基本である。その後の改良において行番号の代替となるラベルやジャンプ命令の代替となるサブルーチンが導入されたが、1970 年代にダイクストラらにより発展した、現代のプログラミングの基本概念であるプログラムの構造化 [7] さえも素直に記述できないため、現代の教育用言語としての品質は備えていない。

1990 年代に、マイクロソフト社により Windows の GUI 環境に対応した VisualBasic [63] が開発され、コンパイラによる実行オブジェクトの生成や、MicrosoftOffice 等のマクロ言語としての利用が可能になった。VisualBasic では、あらかじめ用意された GUI 部品を中心としたプログラム部品 (カスタムコントロール) を組み合わせるプログラミングが可能になった。また、関数の定義が可能になるなど基本的な言語構文が整えられた。一方、必要な機能があらかじめ用意された部品に存在しないときは、部品そのものを自分で開発する必要がある。汎用的な部品を開発する作業は一般に高度なプログラミングを必要とするため、初心者にとって困難な作業になってしまうという問題がある。

BASIC は 1993 年に「電子計算機プログラム言語 Full BASIC」[66] として日本工業規格 (JIS) で規格化された。構造化や例外処理など基本的な機能が追加されたが、行番号が残るなど言語仕様としては古典的な言語から発展していない。

図 2.6、図 2.7 に、JIS Full BASIC のプログラム例を示す。

```
10 PRINT "Hello World"
```

図 2.6: JIS Full BASIC のプログラム例 (Hello World)

このプログラムを入門用のサンプルとして使うことを考えると、次の問題があることがわかる。

1. 構造化されていない

   プログラムの構造は、行番号を単位としたジャンプ命令を基本とする。その後、ラベルやサブルーチン呼び出しが導入されたが、関数やブロックの概念がないため、呼び出しや変数の範囲がプログラム全体になってしまい、常にプログラム全体を把

```
10 INPUT N
20 LET S=0
30 FOR K=1 TO N
40 LET S=S+K∧2
50 NEXT K
60 PRINT S
70 END
```

図 2.7: JIS Full BASIC のプログラム例 (数の計算)

握してプログラミングを行う必要がある。初心者の入門用に適さないことはもちろん、実用的なプログラム開発にも適していない。

白石 [55] [56] は、教育用の言語としてマイクロソフト系の独自仕様の BASIC を採用することの問題点を次のように指摘している。

- 比較演算の結果が数値であるため、「`20 <= N < 30`」という式の結果が通常の数学の意味と異なってしまう。
- 数値に複数の型があり、使い分けが難しい。
- 例外処理が `GOTO` によるジャンプ命令しかない。

BASIC は 1980 年代に入門用に使われた時期があったが、言語仕様が古く表現力に乏しいことから、現在の学校教育での利用には適していない。

LOGO

LOGO [26] は、学校教育における利用を意図して 1967 年に Papert により設計された言語である。

LOGO は図形や数学的知識の発見的な学習を目的としており、プログラミングの学習を直接の目的としていない。しかし、学習の過程でプログラミングを利用するために、言語の習得に負担が小さくなるよう工夫されている。

通常の言語では例題で数値や文字列を扱うのが一般的だが、LOGO の画面には図形や動物の形をしたタートルというオブジェクトが表示され、それを移動した軌跡で図形を描く「タートルグラフィックス」を実現している。小谷 [53] は、タートルグラフィックスでは画面に目に見える対象が表示され、命令を実行するたびに実行の結果が画面に軌跡として反映されるため、プログラムの動作を対話的に確認しながら実行することができる利点を指摘している。図 2.8、図 2.9 に LOGO のプログラム例を示す。

```
PRINT "|Hello World|
```

図 2.8: LOGO のプログラム例 (Hello World)

```
TO SQUARE :R :N
  IF (:N < 1) [STOP]
  FORWARD :R
  RIGHT 90
  SQUARE :R (:N - 1)
END
```

図 2.9: LOGO のプログラム例 (n 角形の描画)

プログラムは FORWARD や RIGHT など、わかりやすい英単語により記述する。英語のほかにも各国語に対応した製品が作られている。国内においても、1980 年代に「前へ」「右へ」のような日本語命令を使える製品が現れたことから、今までに 20 冊以上の入門書 [54] [58] が出版されるなど、広く利用されてきた。

図 2.9 のプログラムを入門用のサンプルとして使うことを考えると、次の問題があることがわかる。

1. 複雑なプログラムに向かない

   LOGO の言語仕様は LISP[33][61] の概念を継承して設計されており、制御構造として再帰が、データ構造としてリストが用意されている。LOGO は少ない概念でプログラムを書けるという利点を持つが、学習者が少しでも複雑なプログラムを記述しようとすると再帰やリストという高度な概念を学習することが必要になってしまう。

図 2.9 のプログラムでは、手続き SQUARE の中で自分自身 (SQUARE) を呼び出す再帰処理を行い、n 角形を描画するためにループを形成している。再帰という高度な考え方を理解しなければならないことに関連して、Bork[3] は、LOGO の授業において、対話的なタートルグラフィックス以上の学習に結び付けるのが難しいという問題を指摘している。また、このプログラムでは、ループの終了条件を条件文で記述しているが、終了条件を誤って記述すると無限ループになりプログラムが終了しない。これは試行錯誤をしながら学習を進める初心者にとって混乱を招く可能性がある。

2. 部品化できない
   LOGO は画面にタートルが表示されることから、本質的にオブジェクト指向の概念と相性がよいことが期待される。しかし、実際には多くの LOGO の言語仕様は提案された当初から拡張されていない。そのため、オブジェクト的なプログラミングを手続き的な操作で記述する必要があり、複数のタートルを扱えるように拡張したり、GUI 部品を扱えるようにするように拡張した場合にプログラミングが難しくなってしまう傾向がある。

3. オブジェクト指向でない
   LOGO の多くの処理系はオブジェクト指向を取り入れていない。LOGO にオブジェクト指向を取り入れた例として ObjectLogo [1] がある。しかし、ObjectLogo ではクラス方式のオブジェクト指向を採用しているため、プログラムの中で事前にクラスを設計し定義する作業が必要になり、LOGO の特徴である対話的な実行を難しくしてしまっている。

LOGO はタートルグラフィックスの採用によって児童教育用途を含む入門用言語の地位を確立したが、言語仕様が古くタートルグラフィックス以上の学習に進むのが難しいことから、現在の学校教育での利用には適していない。

2.3.4 ビジュアル言語

ビジュアル言語 [5] [52] [68] [71] [73] は、図形やアニメーションを用いてプログラムを記述する言語である。文字によるテキスト表現を行わず、画面に表示された図形などを使

用することで、プログラムを視覚的に扱うことができる。

ビジュアル言語ではキーボードの入力を不要にできることから、初心者用のプログラミング用途に使われることがある。マウスを使い画面上でプログラミングを行う言語として SqueakToys[34] などが存在する。しかし、図形的な表現はプログラミングの本質的な難しさを解決するものではない点に注意が必要である。竹内 [62] は、ビジュアル言語はプログラミングの面倒さを解決するが、本質的な部分の困難さを解決するものではないことを指摘している。

ここでは、画面上でプログラミングを行う例と、実世界のブロックを扱う例として、LEGO MindStorms [2] とアルゴブロック [79] を取り上げ検討する。

LEGO MindStorms

LEGO 社の MindStorms [2] は、LEGO ブロックで作ったロボットや車両などを制御するための製品である。

図 2.10 に、CPU などを内蔵した本体である RCX と、それにモーター、センサーなどが接続された様子を示す。図 2.11 はプログラムを行う画面である。パーソナルコンピュータ上でブロックを接続する形でプログラミングを行い、作成したプログラムを本体に転送し実行する。

MindStorms はロボットのような対象を順に操作する手順を記述することを想定した言語を提供しているが、複数のオブジェクトを扱うような汎用のプログラミングを意図したものではない。

MindStorms は自作のさまざまな機能を持つロボットを制御できることから、動作結果の視覚的なフィードバックや学習の動機付けに効果がある。

一方、このプログラムを入門用言語として使うことを考えると、次のような問題がある。

1. 用途が限定されている
   ロボットを制御する目的に限定されているため、汎用のプログラミング学習に利用できない。

2. 複雑な構造を記述できない
   画面に限られたブロックを並べていくため、複雑なプログラムの記述には適していない。

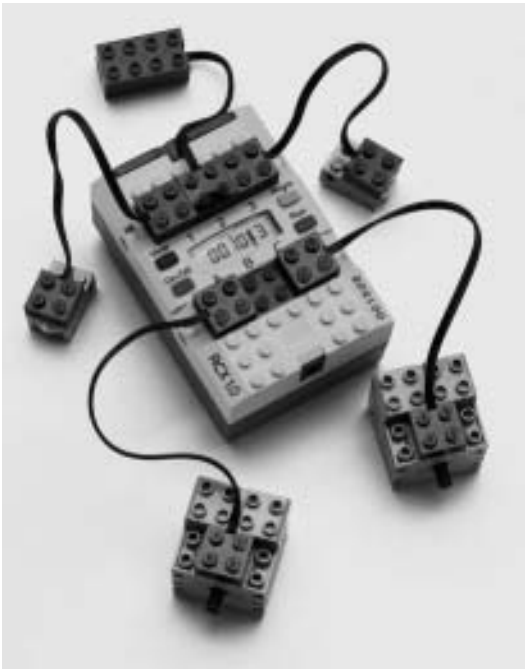


図 2.10: MindStorms の制御装置

図 2.11: MindStorms のプログラミング画面

MindStormsは、モーターなどを制御するプログラミングを手軽に体験することには有効であるが、学校教育の中で汎用的な用途に使えるものではない。

アルゴブロック

アルゴブロック [79] は、ブロックを並べることにより、プログラミングを体験できるようにした製品である。

図 2.12 のブロックは、それぞれが「前へ 5 歩進む」「右へ 90 度曲がる」「ここから 4 回繰り返し」「ここまで繰り返し」などの意味を持ち、プログラム部品の役割をする。これらのブロックを並べることにより、ディスプレイ上にある対象物の動きを制御し、たとえば「障害物を避けながらスタートからゴールに移動させる」といったプログラミングを行う。

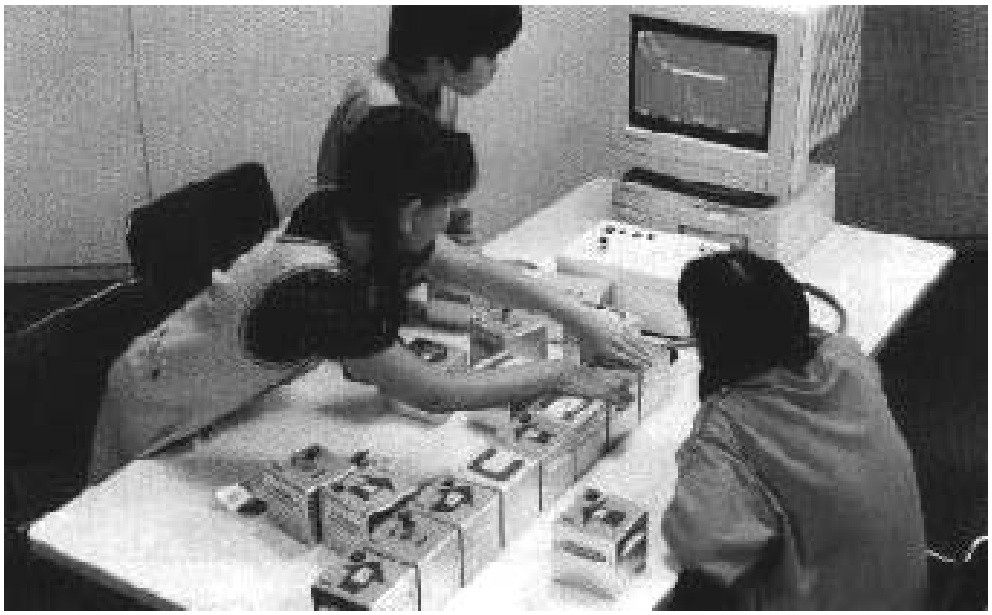


図 2.12: アルゴブロック

限られたスペースで比較的単純な機能をもつブロックを組み合わせることから、複雑なプログラムを作ることは難しいが、手で触れることができるプログラミング環境として、画面の中で記述する言語とは違った効果が期待できる。

このプログラムを入門用言語として使うことを考えると、次のような問題があることがわかる。

1. 複雑な構造を記述できない
   限られた数のブロックを並べていくため、複雑なプログラムは記述できない。

2. 構造化が行えない
   逐次処理や単純な繰り返しといった、フローチャートレベルの低い制御構造しか表現できない。

アルゴブロックは、計算機の逐次実行や単純なアルゴリズムを学ぶことに使用できるが、汎用的なプログラミングを行えるものではないため、学校教育の中で汎用的に使えるものではない。

2.3.5 例示プログラミング

例示プログラミング[1] [11] [27] [28] [29] は、計算機に行ってほしい動作を例で示すことにより、間接的にプログラミングを行う方式である。

プログラミングを行うために計算機言語の文法を覚えなくてよいことが利点であるが、実際には計算機に例示内容を正確に伝えるための工夫が必要であり、例外を含めたさまざまな例を繰り返し示さなければならないといった問題点がある。

また、例示により計算機が期待したとおりの動作を行うようになったときも、裏でどのような学習やアルゴリズムが生成されたかが見えないため、計算機の動作原理の理解には結び付きにくい。

以下では例示プログラミングの例として、論文が公開されている KidSim [4] と Gamut [23] を取り上げ検討する。

KidSim

KidSim [4] は、画面に表示されたグリッド上に配置された物体を動かして示すことにより、物体の移動規則を例示する。結果として、画面上の対象物が移動するアニメーションを表現することができる。図 2.13 に実行の様子を示す。

[1] PBE (Programming By Example) または PBD (Programming By Demonstration) と呼ばれることもある。

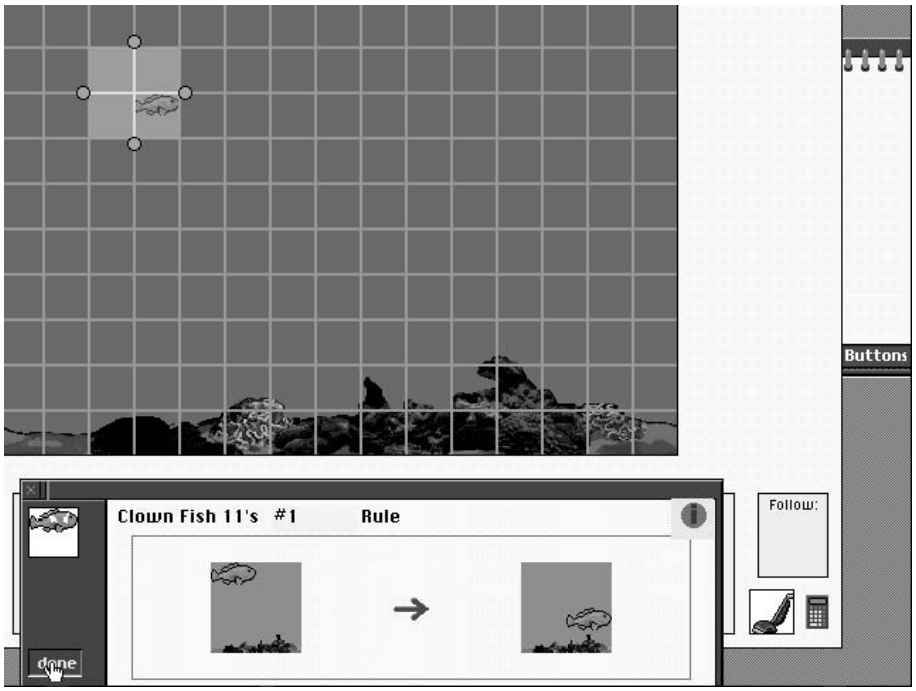


図 2.13: KidSim の実行画面

このシステムをプログラミング教育用の手段として使うことを考えると、次のような問題があることがわかる。

1. プログラムを記述できない
   プログラムを例として示すことはできるが、プログラムを明示的に記述することはできない。

2. プログラムの動作原理につながらない
   計算機が内部でどのように実行しているかが見えないため、計算機の動作について理解が深まらない。

KidSim は、画面上でキャラクタを動かすアニメーションを容易に作ることができる。一方、汎用的なプログラムを記述するための環境ではないため、学校教育の中で汎用的に使えるものではない。

Gamut

Gamut [23] は、例示によりゲームやシミュレーションを記述するプログラミング環境である。

ユーザは例を示すだけでなく、推論に積極的に関与することが可能である。たとえば、実行時に見えなくなるような線で推論のヒントを与えたり、誤った動作をしたときに知らせる、動作すべきときに何もしなかったらそれを知らせるなどを記述できる。オブジェクトの過去の状態を見せるためにオブジェクトを半透明で見せる機能などが用意されている。図 2.14 に実行の様子を示す。

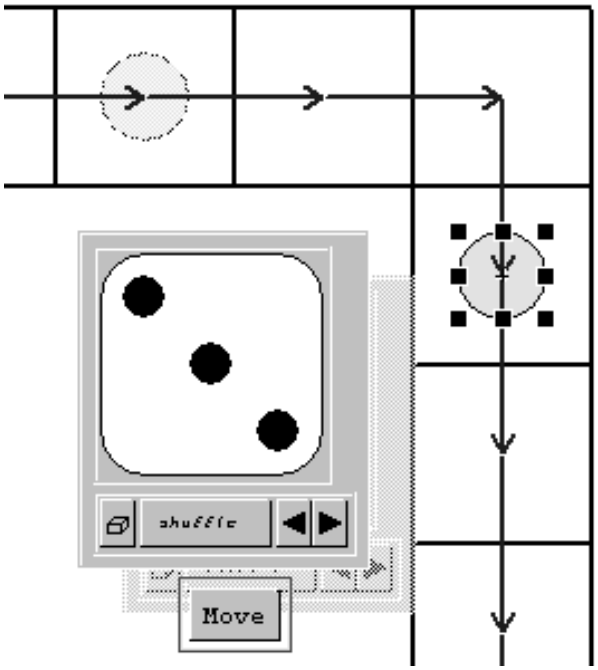


図 2.14: Gamut の例示画面

このシステムをプログラミング教育用の手段として使うことを考えると、次のような問題があることがわかる。

1. プログラムを記述できない
   プログラムを例として示すことはできるが、プログラムを明示的に記述することはできない。

2. プログラムの動作原理につながらない

計算機が内部でどのように実行しているかが見えないため、計算機の動作について理解が深まらない。

Gamut は、計算機に行わせたい動作を示していくことで、プログラミングに相当する行為を行うことができる。一方、明示的にプログラミングを行うための環境ではないため、学校教育の中で汎用的に使えるものではない。

2.4 情報教育に適したプログラミング言語に関するまとめ

本章では、教育用途で使われることのあるプログラミング言語を 5 つのグループに分類した後、評価のための基準を定め、言語の評価を行った。

表 2.1 に、本章で検討した既存言語の評価をまとめた。全体として、機能の高い言語は理解することが難しく、入門用の言語は限定された機能しか扱うことができないという傾向が存在する。本節までの検討から、「学校教育の中で容易に扱うことができ、必要な機能を満たす」言語が存在しないことを示した。

オブジェクト指向は現在のソフトウェアでは重要な役割を果たしており、実用的なソフトウェアを効率よく体験するための効果的な手段である。しかし、従来の言語では、オブジェクト指向を扱える言語は学習の難しい言語に限られており、多くの入門用の言語ではオブジェクト指向を取り入れることができていない。

本論文で提案するプログラミング言語「ドリトル」は、「オブジェクト指向を扱うことができ、学習が難しくない」という学校教育に適した特徴を持つ言語である。次章以降では、ドリトルの設計と各種評価について述べる。

表 2.1: 言語の比較

	(1) 理解しやすさ	(2) 対話性	(3) 構造化	(4) オブジェクト指向	(5) ネットワーク
(汎用言語)					
Java [13]	×	×	○	○	△
C++ [35]	×	×	○	○	△
(高等教育用言語)					
Blue [20]	×	×	○	○	×
若葉 [48]	○	○	○	×	×
(教育・入門用言語)					
LOGO [26]	○	○	○	×	×
BASIC[56]	○	○	×	×	×
(ビジュアル言語)					
MindStorms [2]	○	○	×	×	×
アルゴブロック [79]	○	○	×	×	×
(例示プログラミング)					
KidSim [4]	○	○	×	×	×
Gamut [23]	○	○	×	×	×

○: 項目を満たす。

×: 項目を満さない。

△: 可能だが、初心者には容易に使えない。

第3章　プログラミング言語「ドリトル」の設計

第2章では、情報教育に適したプログラミング言語の性質を検討した。本章では、筆者が開発したプログラミング言語「ドリトル」について述べる。ドリトルは学校教育での利用を想定して設計されており、画面上に表示されたオブジェクトにメッセージを送ることでプログラムを記述する。動物やボタンなど実世界の「もの」に相当する「オブジェクト」を単位としてプログラムを書くことで、プログラミングの初心者や生徒が比較的人間に近い考え方でプログラムを記述できることが特徴である。

3.1　プログラミング言語「ドリトル」の設計方針

本節ではドリトルの設計方針を示す。表3.1に、第1.3節で挙げた教育用言語に望まれる性質とドリトルの設計方針との対応を示す。

表 3.1: 望まれる性質と設計方針の対応

教育用言語に望まれる性質 (第1.3節)	ドリトルの設計方針
理解しやすく習得の時間が短い	簡潔な構文
	日本語との対応
動かしながら段階的に学べる	インクリメンタルなプログラミング
基本的な計算機の原理を学べる	テキストによる記述
	アルゴリズムと構造化
日常使うソフトウェアの原理に結び付く	オブジェクト指向
	プロトタイプ方式
ネットワークを体験できる	ネットワークとの通信

以下に、ドリトルの設計方針を説明する。

3.1.1 簡潔な構文

階層的な構文を避けて理解しやすい構文にする。従来のプログラミング言語においては、構文の入れ子構造は当然のこととして受け入れられて来たが、変数の有効範囲などが階層的になり、初中等教育では理解が難しい。従って、言語仕様としては複数レベルの入れ子が書けるとしても、2レベル以上の入れ子構造を作る代りに内側の動作を別のメソッドとして分離して入れ子のレベルを抑えるようなプログラミングスタイルを採れる言語とする。

3.1.2 日本語との対応

日本語との対応性を考慮する。初等（小学校）からの利用を考えると、英語の使用を前提にすることはできない。よって、英語の予約語等は一切用いないものとする。名前への日本語の利用だけでなく、基本的な記号（かっこ、ピリオド等）を含め、日本語の文字だけで記述可能な言語とする。さらに、かぎかっこ ([ ]) でなく日本語の括弧（「 」）を許すこととする。語順も可能な範囲で日本語の語順に近づける。

3.1.3 インクリメンタルなプログラミング

インクリメンタルなプログラミングに対応する。従来BASICやLOGOが教育用に使われてきた要因の1つとして、文を1つだけ打ち込んでも動作が観察できるという点が挙げられる。この利点を引き継ぐために、常識的な長さの1行だけでそれなりの動作が記述でき、それがそのままメソッドとしても定義できることが必要と考える。「クラスの中にメソッドがあり、メソッドの中に文がある」という旧来のクラス方式のオブジェクト指向言語は、複数行を単位として考える必要があり、学習の難易度が高いという問題がある。

3.1.4 テキストによる記述

テキスト表現に基づくソースコードを記述する。近年、例示プログラミングや図的プログラミングのように、画面上で図形的な操作を行うことでプログラミングを行うアプローチがある。しかし、「人工言語として決まった記法による表現を解釈実行する」という計算機の本質を体験する目的から、テキストに基づくソースコード表現を打ち込んで動かすというモデルを扱う。

3.1.5 アルゴリズムと構造化

アルゴリズムを記述できるようにする。数値、文字列、配列等の基本データを扱い、繰り返しや条件分岐等の制御構造を備え、手続きに相当するメソッドを定義することで、構造化されたアルゴリズムを記述できるようにする。この性質により、CPU の演算や多くのプログラミング言語で扱われる計算機の基礎概念を体験することができる。

3.1.6 オブジェクト指向

オブジェクト指向言語とする。現代のソフトウェア開発においては、オブジェクト指向機能を持つ言語を採用し、既に用意されている部品を再利用することで短期間に高度な機能を持つソフトウェアが作成できるようになっている [12]。限られた授業時間で小中学生を含む生徒がソフトウェアを作ったという達成感を感じるために、この性質は教育用の言語にも必要である。

3.1.7 プロトタイプ方式

プロトタイプ方式とする。現代において多くのオブジェクト指向言語はクラス方式であるが、クラス方式ではクラス、インスタンス、継承など理解しなければならない概念が多くなり、「敷居の低さ」の点で問題があると考える。プロトタイプ方式 [6] [37] であれば、あるオブジェクトのコピーは元のオブジェクトの性質を引き継いでいる、という常識的なパラダイムのみで済むので、より教育用として適していると考える。

3.1.8 ネットワークとの通信

インターネットや携帯電話を含む電子メールの普及により、計算機の活用がネットワークと切り離せないものになってきている。ネットワークとの通信を行うことで、画面の中で動くだけのプログラミング学習を超えて、制御やネットワークの学習にも発展することができる。

3.2 言語の設計

図 3.1 に、拡張 BNF[1]記法 (表 3.2) で記述されたドリトルの構文を示す。一般的なプログラミング言語と比較して、記号の少ない簡潔な記述が可能であり、識別子に日本語文字を使用できるという特徴をもつ。

表 3.2: 拡張 BNF 記法

構文	意味
::=	左辺を右辺で定義する
'X'	X という文字
[X]	X は省略可能
(X Y)	X と Y の並び (グルーピング)
X | Y	X または Y
X…	X の 0 回以上の繰り返し
< X >	その概念の解説文

ドリトルのプログラムの動作は「式」の集まりとして記述される。「式」は 1 つ以上のメッセージ送信を組み合わせたものである。基本的なメッセージ送信は

[1]Backus-Naur Form。プログラミング言語 Algol60 の構文を記述するために用いられた。拡張 BNF はこれを拡張した記法である。たとえば図 3.1 の次の定義

中置式 ::= 中置式 演算子 中置式 | 項

では、項が数値を含む構文の定義であり、中置式が四則演算の演算子を含む構文の定義であることから、「3 + 4」「2」はいずれも中置式である。また、BNF の右辺に左辺の記号を置くことで、再帰的な定義を記述できる。たとえば「3 + 4」「2」という中置式同士に演算子を適用することで、「3 + 4 - 2」も中置式と解釈される。

```
プログラム ::= (文 '。')…
文 ::= 代入文| メソッド定義| 式
代入文 ::= 変数 '=' 式
メソッド定義 ::= 変数 '=' ブロック
変数 ::= [項 ':'] 識別子
式 ::= 単純式 | メッセージ送信
メッセージ送信 ::= [レシーバ] '!' メッセージ
レシーバ ::= 項
メッセージ ::= 引数… メソッド名 (';'] 引数… メソッド名)…
引数 ::= 単純式
メソッド名 ::= 識別子
括弧 ::= '(' 中置式 ')' | '(' メッセージ送信 ')'
単純式 ::= 数値リテラル|文字列リテラル|括弧|ブロック
ブロック ::= '「' ['|' 識別子… '|'] 文 ('。' 文)… '」'
中置式 ::= 中置式 演算子 中置式 | 項
項 ::= 単純式 | 識別子
演算子 ::= '+' | '-' | '*' | '/' | '>' | '<' | '>=' | '<=' | '==' | '!='
文字列リテラル::= '"' <引用符以外の任意の文字>… '"'
  | '『' <引用符以外の任意の文字>… '』'
数値リテラル ::= ['±'] 数字… ['.' 数字…]
識別子 ::= (英字|仮名|漢字)(英字|仮名|漢字|数字)…
```

図 3.1: ドリトルの構文

オブジェクト!引数 引数 … 識別子

という形をしており、「!」の前に指定したオブジェクトの「識別子」で表されるメソッドを実行させる。

数値や「(…)」で囲まれたものは引数として扱われ、そうでない最初の識別子がメソッド名となる。メソッドはオブジェクトを返すことができ、そのオブジェクトに対してさらにメソッド呼び出しを行うことができる。これらを合わせるとより一般的なメッセージ送信は

オブジェクト!引数… 識別子 引数… 識別子 … 識別子

の形になる。

数値もオブジェクトであり、メッセージ送信記法によって演算を指定できるが、それでは読みにくいため通常の中置記法も使うことができる。

1 ＋ 2

はメッセージ送信記法

1！2　足す

と同等に扱われる。1つの式は全体としてメッセージ送信式か中置記法の式かのいずれかでなければならないが、「( )」の中はそれぞれ独立に記法を選ぶことができる。従って、たとえばメッセージ送信式の中に中置記法を混ぜたければ「( )」で囲んで記せばよい。また、メッセージ送信式の中では識別子はメソッド名として扱われるので、変数(プロパティ)を参照したければ同様に「( )」で囲んで記す。この場合、「( )」の内側は中置記法として扱われ、中置記法では識別子は変数(プロパティ)の値を参照する。

図3.2に、ドリトルで記述したHello Worldのプログラム例を示す。2.3節で紹介した各言語と比較して、Java、C++、Blueと同様のオブジェクト指向言語でありながら、若葉、BASIC、LOGOなどの入門用言語と同等の簡潔な記述が可能である。

```
ラベル！『Hello World』 作る。
```

図 3.2: ドリトルのプログラム例 (Hello World)

図3.3にフランスの国旗を描くサンプルプログラムを示す。以下に、サンプルプログラムの解説をまじえながら言語の主要な機能と特徴を解説する。

```
カメ太=タートル！作る。
矩形=カメ太!50 歩く 90 左回り 100 歩く 90 左回り 50 歩く 閉じる 図形にする。
矩形！（青）塗る。
白矩形=矩形！複製 50 0 移動する（白）塗る。
赤矩形=白矩形！複製 50 0 移動する（赤）塗る。
```

図 3.3: プログラム例（フランス国旗）

3.2.1 メッセージ送信

ドリトルはオブジェクト指向言語であり、オブジェクトへのメッセージ送信が基本となる。「!」の左はレシーバであり、右にメソッド名を書く。

```
カメ太 = タートル！作る。
```

この例では「タートル」は広域変数名[2]（広域変数はトップレベルオブジェクト「ルート」のプロパティとして格納されている）であり、そこには予め用意されたタートルオブジェクトが格納されている。

「作る」メソッドは新しいオブジェクトを生成し、そのプロトタイプをタートルオブジェクトにする。これにより、以後このオブジェクトはタートルオブジェクトのプロパティおよびメソッド一式と同じものを予め持つかのようにふるまう。

[2]グローバル変数とも呼ばれる。対になる概念はローカル変数である。

「カメ太」はここで新たに作成する広域変数名であり、そこに新しく作ったオブジェクトを格納する。ドリトルでは変数（オブジェクトのプロパティ）は最初に値を格納した時に作られ、予め宣言する必要はない。

実行の結果として、画面には複製されたタートルオブジェクトが表示される。タートルオブジェクトの初期状態は、画面の中央に存在し、X 軸方向 (右方向) を向いている。

3.2.2 メッセージ送信のカスケード

メッセージは任意個数の引数を持てる。引数のうち数値リテラルや文字列リテラルはそのまま書けるが、変数参照や一般の式は「( )」で囲む必要がある。識別子が来るとそれがメソッド名として認識され、そこまでが 1 つのメッセージ送信式となる。

メッセージはオブジェクトを値として返すので、その右側に続けて引数とメソッド名を書くことで、返されたオブジェクトに対するメッセージが続けて指定できる。これをカスケード（直列）送信と呼ぶ。カスケード送信はいくつでも書くことができ、「。」でその最後を表す。

矩形=カメ太!50 歩く 90 左回り 100 歩く 90 左回り 50 歩く 閉じる 図形にする。

この例では次のステップで実行が行われる。

(1) 「カメ太！50 歩く」が実行され、返値としてカメ太自身が返る

(2) カメ太に「90 左回り」が送られ、返値としてカメ太自身が返る

(3) カメ太に「100 歩く」が送られ、返値としてカメ太自身が返る

(4) カメ太に「90 左回り」が送られ、返値としてカメ太自身が返る

(5) カメ太に「50 歩く」が送られ、返値としてカメ太自身が返る

(6) カメ太に「閉じる」が送られ、返値としてカメ太自身が返る

(7) カメ太に「図形にする」が送られるが、このメソッドはカメ太ではなくこれまでに描かれた軌跡を新たな図形にしたものを返す

(8) 新しく作られた図形オブジェクトを広域変数「矩形」に格納する

メッセージを「;」で区切ると、メッセージは直前のメッセージ実行で返されたオブジェクトではなく、「!」の左に書かれたレシーバに対して送られる。

```
カメ太!図形にする; 90 左回り。
```

この例では、カメ太に「図形にする」が送られて図形オブジェクトが返されるが、その返値は使われず、「90 左回り」がレシーバである「カメ太」に送られる。

3.2.3 リテラルと変数の参照

数字で始まるトークンは数値リテラルである。リテラルの末尾に単位をつけることができる[3]。文字リテラルは文字を「" "」または「『 』」で囲んだものとして表す。

```
矩形！(青) 塗る。
```

それ以外のリテラルは用意していないが、色などはその色名に対応する広域変数に値を格納しておくことでプログラム作成者の便宜を計る。たとえば、広域変数「青」には青色を表す色オブジェクトが格納されている。ここでは、「( )」の中に書くことでメソッド名ではなく変数への参照を表している。

3.2.4 オブジェクトの複製

「複製」メソッドはレシーバオブジェクトの複製を返す。複製されたオブジェクトは元のオブジェクトと同じプロパティ、メソッド、プロトタイプを持つ。

```
白矩形=矩形！複製 50 0 移動する (白) 塗る。
赤矩形=白矩形！複製 50 0 移動する (赤) 塗る。
```

上の例では矩形オブジェクトを複製し、位置を移動した後で色を塗っている。結果として、フランスの国旗が描かれる。

[3]現在の仕様では、単位はコメントとして扱われる。

3.2.5　メソッド 定義とブロック

メソッドはオブジェクトのプロパティとしてブロックを格納することで定義する。ブロックは [⋯] または「⋯」で表し、その内側のコード列は後でブロックが評価される時に実行される。

カメ太：矩形描画 ＝
「|x y|！(x) 歩く 90 左回り (2 ∗ x) 歩く 90 左回り (x) 歩く 閉じる (y) 塗る」。

ブロックは任意個数のパラメタを持つことができる。パラメタはブロックの先頭に「| 識別子⋯|」という形で指定する（指定がない場合は引数のないブロックとなる）。

パラメタはブロックの実行中だけ存在する無名のコンテキストオブジェクトのプロパティとして扱われ、初期値としてブロック評価時に渡された実引数値が格納されている。

ブロックを格納しているプロパティはメソッドとして実行可能である。その際、メソッド名より前に書かれた実引数値が 1 つずつパラメタに対応する。実引数の数がブロックが持つパラメタより多い場合はそれは捨てられ、少ない場合はパラメタは未定義オブジェクトを初期値として持つ。ブロック内で最後に評価された式の値がメソッドの返値となる。

一部のオブジェクトでは、特別な名前のメソッドを定義しておくことにより、特定の条件が成立したときにメソッドを実行することができる。

カメ太：衝突 ＝ 「！150 右回り」。

この例では、カメ太という名前のタートルオブジェクトに、他のオブジェクトと重なったときに実行されるメソッドを定義している。「衝突」メソッドはタートルオブジェクトと図形オブジェクトに定義可能であり、他のタートルオブジェクトまたは図形オブジェクトと領域が重なったときにメソッドの実行が行われる。

「!」の左のレシーバを省略した場合には、実行時のコンテキストにおけるレシーバにメッセージが送られる。よって、カメ太が画面上で他のオブジェクトと衝突した場合には、カメ太に対して「150 右回り」が送られる。

前進ボタン：動作 ＝ 「カメ太！100 歩く」。

この例では、前進ボタンというボタンオブジェクトに、ボタンが押されたときに実行されるメソッドを定義している。ボタン自身ではなく他のオブジェクト (カメ太) にメッセージを送るために、「!」の左でレシーバ「カメ太」を指定している。

3.2.6 条件判断

大小比較などの論理演算子は論理値オブジェクトを返す。制御構造は、論理値オブジェクトを返すようなブロックオブジェクトへのメッセージおよびそのカスケード送信として実現する[4]。

「x>y」! なら 「...」 実行。
「x>y」! なら 「...」 そうでなければ 「...」 実行。

上の例では、ブロックに「なら」を送るとブロック自身を実行し、最後の評価値の真偽に応じて True または False を返す。

オブジェクト True は「ブロック 実行」を送られるとブロックを実行し、未定義オブジェクトを返す。「ブロック そうでなければ」を送られるとブロックを実行し、オブジェクト Done を返す。つまり True は「条件が真だったので then 側へ行く」という状態を表している（Done の意味は後述）。

オブジェクト False は「ブロック 実行」を送られるとブロックを実行せずに未定義オブジェクトを返す。「ブロック そうでなければ」を送られるとオブジェクト True を返す。この True に「ブロック 実行」が送られることで最初の条件が偽だったときに 2 番目のブロックが実行される。一般に False は「まだ条件が真のものはないので else 側へ行く」という状態を表している。

オブジェクト True にはさらに「ブロック なら」を送ることもでき、その（ブロックが返す）論理値が真ならオブジェクト True、偽ならオブジェクト False を返す。これによって次のような if-then-else if が使えるようになる。

「x>y」! なら 「...」 そうでなければ 「x<y」 なら 「...」そうでなければ 「...」 実行。

最後に、Done は「ブロック なら」および「ブロック そうでなければ」に対して Done 自身を返し（ブロックは評価しない）、「ブロック 実行」に対してブロックは実行せず未定義オブジェクトを返す。すなわち Done は「既に then の枝を実行ずみ」という状態を表している。

[4]条件部はブロックに入れずに条件式を直接書くことが可能だが、実行時に必ず評価される点に注意が必要である。ループの終了条件のように繰り返し評価する必要がある場合や、条件分岐の else-if 部の条件のように必要になるときまで評価したくない場合があるので、標準ではブロックの使用を推奨している。

3.2.7　条件による繰り返し

条件式を含むブロック B1 にメッセージ「の間」を送ると、内部にブロックを包含するオブジェクトが返される。

```
「x<100」! の間「カメ太！1 歩く。x= (x+1)」実行。
```

このオブジェクトに繰り返し実行されるコードを含むブロック B2 を引数としてメッセージ「実行」が送られると、オブジェクトは B1 を実行し、その値が論理値の真の間だけ B2 を繰り返し実行する。

3.2.8　タイマーによる繰り返し

タイマーは、ブロックを一定間隔で実行するオブジェクトである。内部に実行間隔と実行時間の状態を持つ。

```
時計 = タイマー！作る 1秒 間隔 10秒 時間。
時計 !「矩形！36度 回転」実行。
時計！待つ。
```

ブロックは非同期に実行され、ブロックを起動した処理はブロックの終了を待たずに先に進む。1 つのタイマーに複数の処理を頼んだときは、タイマーはそれらを直列に実行する。タイマーの実行完了を待ちたければ、タイマーのメソッド「待つ」により同期を取ることができる。

時間の代りに回数を指定できる。回数を「1 回」とすれば、起動した処理と指定したブロックの並行実行の機能としても使うことができる。

3.2.9　イベント 検知

「ボタンが押された」「タートルが他のオブジェクトと衝突した」などのオブジェクトに対する環境の変化は、オブジェクトにメッセージとして通知される。特定の名前のメソッド (ボタンオブジェクトの「動作」やタートルオブジェクトと図形オブジェクトの「衝突」) を定義することで、そのオブジェクトに対するイベントを受け取ることができる。

```
実行ボタン ＝ ボタン！”実行” 作る。
実行ボタン:動作 ＝ 「三角形！ 36 ぐるぐる」。
```

この例では、「実行ボタン」という名前のボタンオブジェクトを作り、押されたときの動作を定義している。

```
カメ太 ＝ タートル！ 作る。
カメ太:衝突 ＝ 「！ 5 戻る 160 右回り」。
```

この例では、「カメ太」というタートルオブジェクトを作り、他のオブジェクトに衝突したときに実行される動作を定義している。

3.2.10 外部機器との通信

シリアルポートオブジェクト[5]を使うことで、外部機器にデータを出力して制御等に利用することが可能である。

```
シリアルポート:前進＝「|x|！ 2 出力 0.1 待つ (x) 出力 0.1 待つ」。
```

この例では、0.1 秒の間隔を空けながら、外部機器に値「2」と引数で受け取った値 (x の値) を出力する「前進」というメソッドを定義している。外部機器は装置の種類によって受け取るデータ形式が異なる。教員が装置に応じた操作命令をドリトルの初期化ファイルに定義しておくことで、生徒は「前進」等の命令を定義せずに使うことができる。

3.2.11 サーバーとの通信

サーバーオブジェクトを使い、ネットワークを介してドリトル用のサーバーと通信することができる。詳細は第 6 章で説明する。

```
サーバー！ ”s00” 接続。
カメ太 ＝ サーバー！ ”カメ太” 複製。
カメ太！ 100 歩く。
```

この例では、「s00」という名前のサーバーに接続し、サーバーに登録されている「カメ太」というオブジェクトをドリトルに複製して操作する。

[5]名前が示すように、このオブジェクトでは RS-232C に代表されるシリアル通信を扱う。この他に、パラレルポートなど操作対象に応じたオブジェクトを用意していく予定である。

3.2.12 標準オブジェクト

言語を学ぶための題材としての役割に加え、オブジェクト指向言語の効果を知ってもらい、また自分のプログラム作成に達成感を持ってもらうためにも、標準オブジェクトとして何を用意するかは重要な問題である。表 3.3 に現時点での標準オブジェクトの一覧を示す。この中で、数値、論理値、文字列、ブロックは言語の基本部分を構成する基本オブジェクトであり、それ以外はシステムが用意する標準オブジェクトである。

表 3.3: 標準オブジェクトの一覧

オブジェクト	説明
数値	数値を表し、各種の演算を実現する
	内部では 64bit の浮動小数点で値を保持する
論理値	論理値を表し、論理演算を実現する
文字列	文字列を表し、連結等の操作を実現する
ブロック	メソッドの定義に使われるほか、
	繰り返しなどの制御構造を実現する
配列	集合データを表現する
タートル	ペン先。タートルグラフィックスを描く
図形（パス）	点や線の集合を表す
色	色を表し、三原色からの色の生成などを実現する
ボタン	押したときの動作を設定できる GUI 部品
フィールド	テキストデータを入力・表示する GUI 部品
リスト	複数行のテキストデータを表示できる GUI 部品
選択メニュー	メニューから選択入力できる GUI 部品
ラベル	テキストを表示する GUI 部品
シリアルポート	外部インターフェースと入出力を行う
サーバー	ネットワーク上のサーバーと通信する

3.3 ドリトルの実装

3.3.1 内部構成

ドリトルの処理系は Java 2 [13] により記述されており、Java 2が動くさまざまな環境で動作する。字句解析、構文解析は SableCC [8] による生成コードを利用し、これに加えて意味解析、インタプリタ、ユーザインタフェース、および標準オブジェクト群を実装している。図 3.4 にドリトルの内部構成を示す。

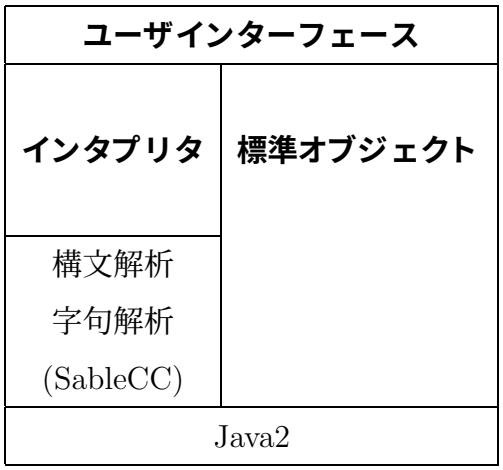


図 3.4: ドリトルの内部構成

表 3.4 に、ドリトル V1.13 のソースコード行数を示す。

表 3.4: ドリトル処理系のソースコード行数

字句/構文記述（SableCC）	160 行
Java 2 コード	5,800 行

3.3.2 ユーザインターフェース

図 3.5〜図 3.11 にドリトルのユーザインターフェース画面を示す。「実行画面 (図 3.5)」「編集画面 (図 3.8)」「マニュアル (図 3.11)」は、画面上部のタブで切り替える。画面下部には「読込」「実行」「中断」「保存」「終了」のボタンがあり、それぞれ「ファイルからプ

ログラム読み込み」「プログラムを実行」「プログラムの実行を中断」「プログラムをファイルに保存」「ドリトルを終了」の機能を提供する。

図 3.5 に実行画面を示す。「実行」ボタンを押すと、画面が実行画面に切り替わり、読み込まれているプログラムの実行を開始する。実行中のプログラムは、「中断」ボタンで停止することができる。

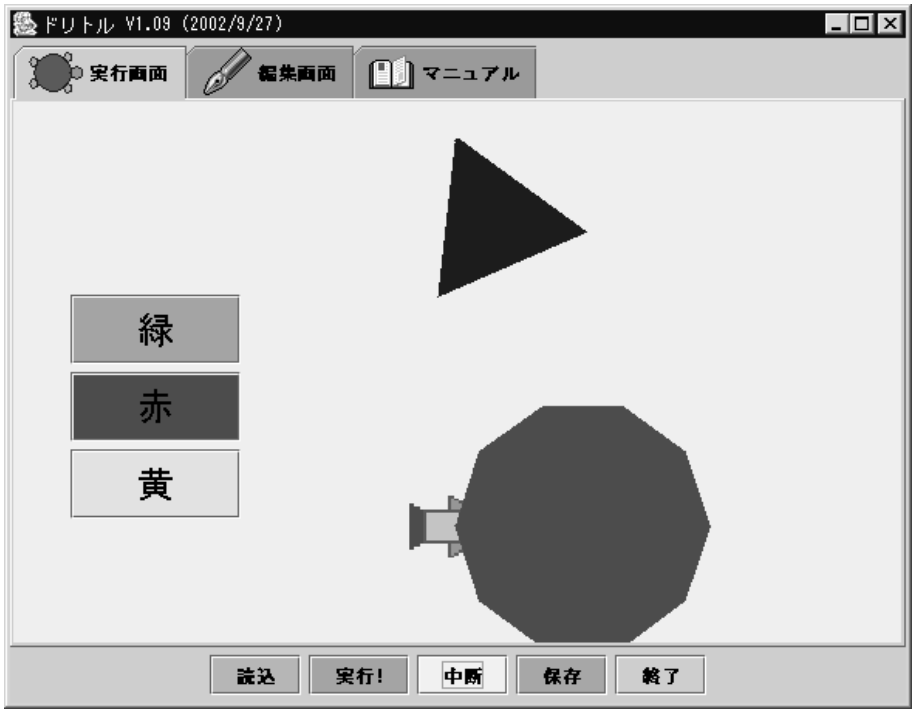


図 3.5: 実行画面

実行時にエラーが発生した場合は、ダイアログでエラーを表示する。図 3.6 に構文エラーの例を示す。

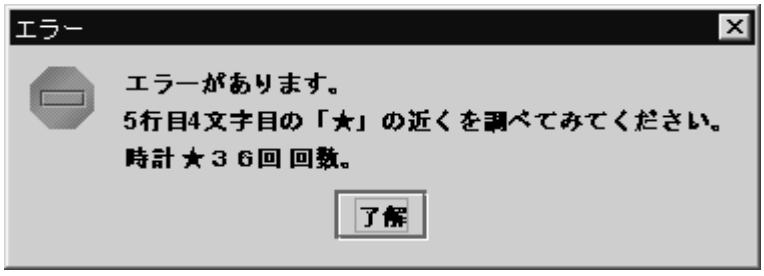


図 3.6: エラー表示 (構文エラー)

この例は

時計！３６回 回数

という構文の「!」を忘れて

時計 ３６回 回数

と記述したときのメッセージである。このエラーは実行ボタンを押した直後に構文エラーとして検出され、ダイアログの中でエラーの位置が行数/桁数と「★」で表示される。また、編集画面を開いたときに、該当個所をカーソルが示す。

図 3.7 に実行時エラーの例を示す。

図 3.7: エラー表示 (実行時エラー)

この例は

カメ太＝タートル！作る。

という構文のメソッド名を入力する際に

カメ太＝タートル！作る a。

と誤入力したときのメッセージである。このエラーはプログラムの実行中にそのメソッドを実行したときに検出され、そのようなメソッドが存在しないことがダイアログに表示される。

図 3.8 に編集画面を示す。この画面ではドリトルのプログラムを入力し、編集を行う。

プログラムはファイルから取り込んだり、ファイルに書き出して保存することができる。図 3.9 にファイルを選択するダイアログを示す。

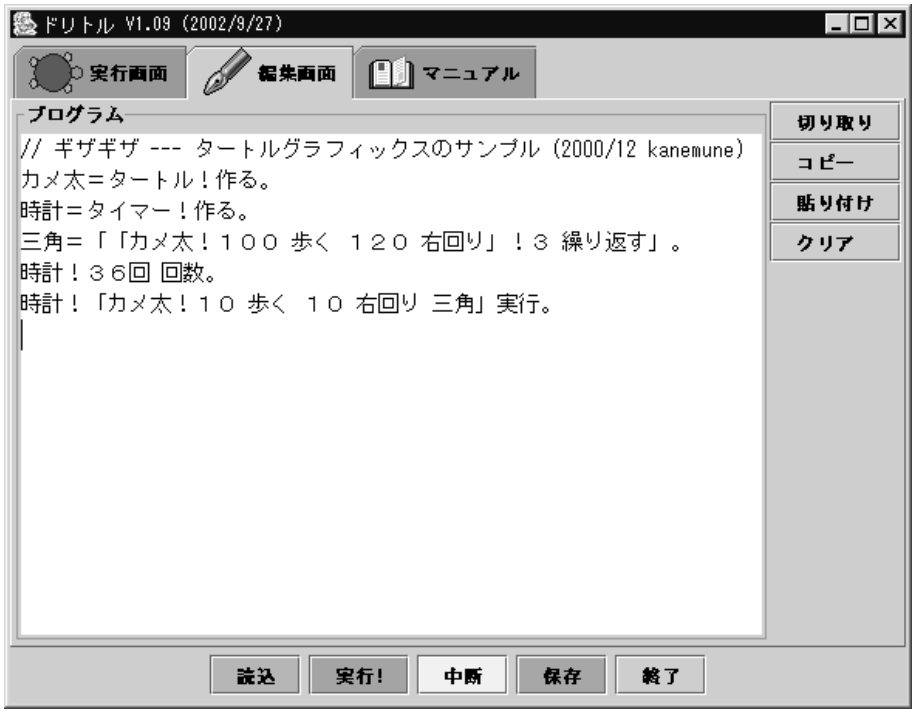


図 3.8: 編集画面

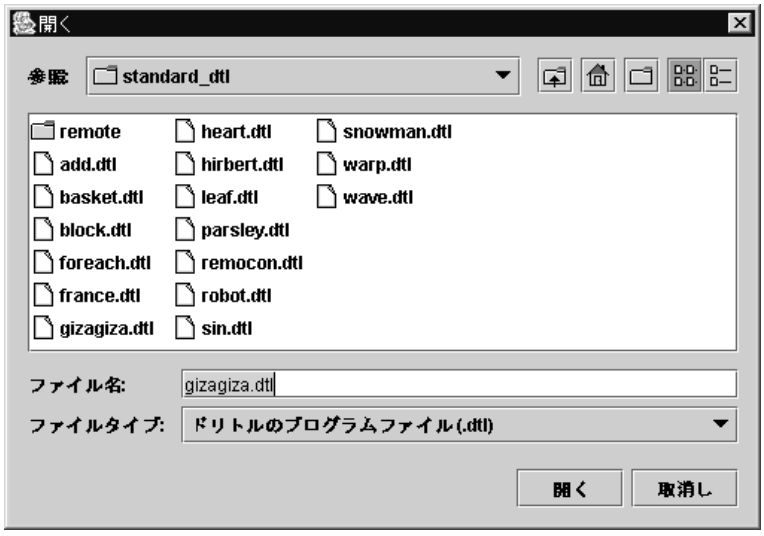


図 3.9: ファイル選択ダイアログ

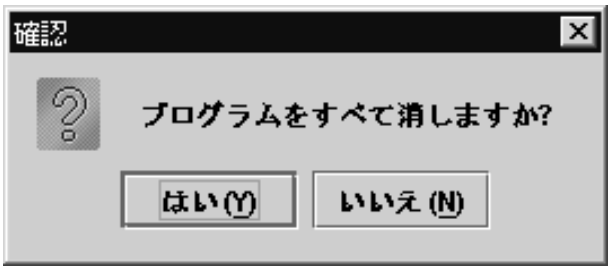


図 3.10: 消去確認ダイアログ

編集画面の右側にはボタンが並んでいる。「切り取り」「コピー」「貼り付け」ボタンはカット&ペーストの機能を提供する。「クリア」ボタンは編集中のプログラムを消去する。消去する前に、ダイアログで確認を行う (図 3.10)。

図 3.11 にマニュアル画面を示す。この画面には、ドリトルに用意されているすべてのオブジェクトのメソッドが表にまとめられており、参照しながらプログラミングを行うことができる。

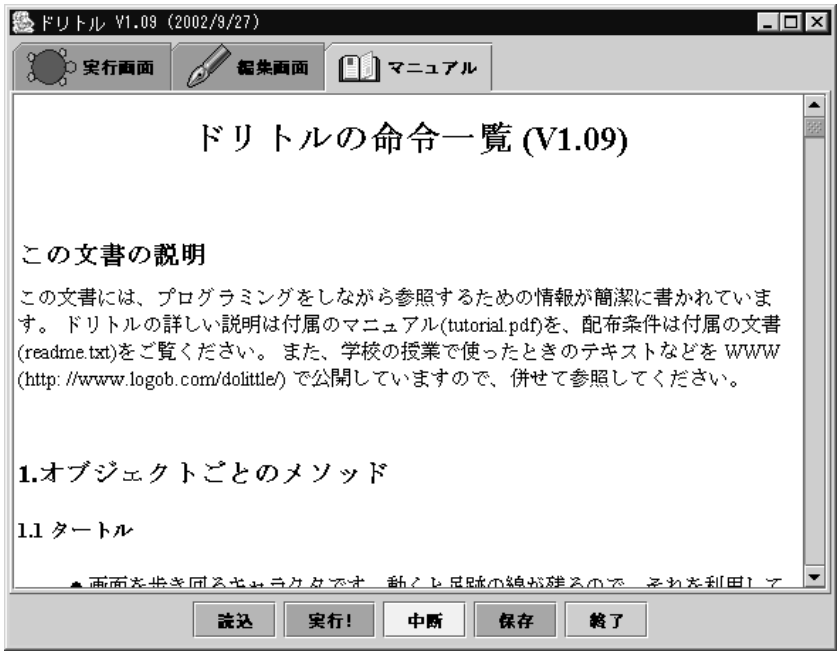


図 3.11: マニュアル画面

第 4 章、第 5 章で述べる中学校等で行われた授業の観察から、ドリトルのユーザインターフェースについて大きな問題はないと考えている。

3.3.3 インタプリタによる実行

実行ボタンによりプログラムの実行が指示されると、現在読み込まれているプログラムを対象として字句解析、構文解析が起動され、エラーなしに抽象構文木が生成された場合にインタプリタによる実行に進む。

インタプリタは抽象構文木を直接たどりながらドリトルの動作を実行する。ドリトルはオブジェクト指向言語であり、プロパティの参照/設定、メソッドの起動など少数の基本動作を除き、ほんとんどの動作はオブジェクトの動作を呼び出すことを通じて行なわれる。

処理系内部では、ドリトルのオブジェクトは Java 言語のクラス OxObject およびそのサブクラスとして実装されている (図 3.12)。クラス OxObject のインスタンスはハッシュ

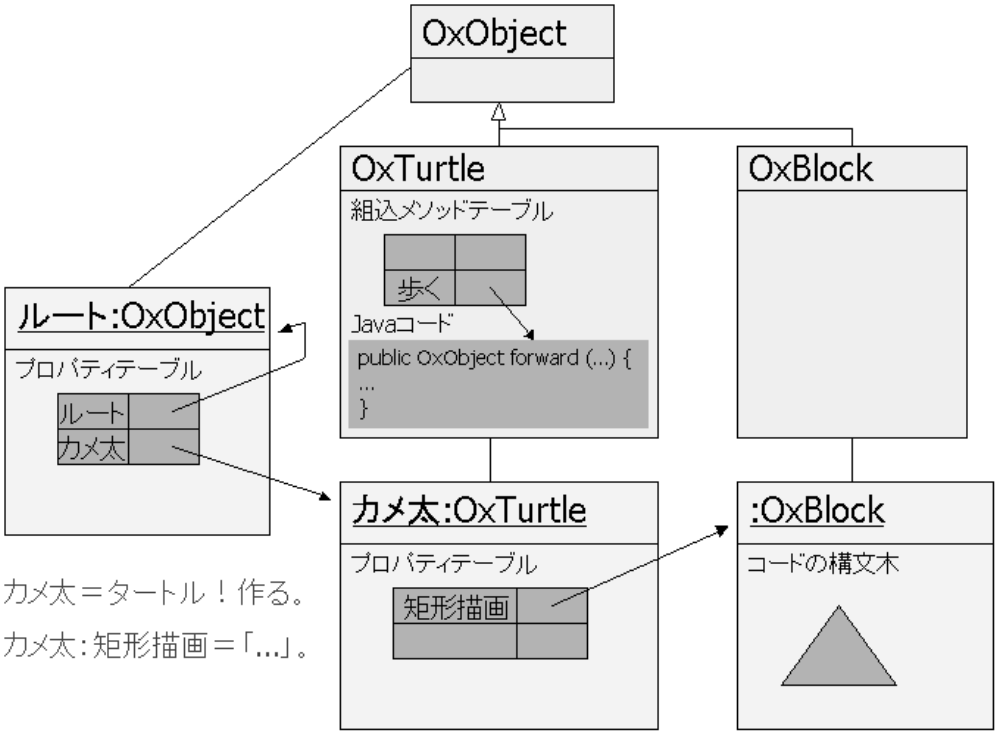


図 3.12: ドリトルの内部クラス

表を保持しており、ここに各オブジェクトのプロパティ値を保管する。また OxObject およびその各サブクラスごとに 1 つずつ別のハッシュ表があり、そこにそのクラスで利用可能な組み込みメソッド（Java で記述されたメソッド）の情報が（Java のリフレクション機能 [13] のメソッドオブジェクトとして）保持されている。

システムに唯一つ存在するルートオブジェクトを除くすべての OxObject およびそのサブクラスのインスタンスは、インスタンス変数 parent に自身のプロトタイプ（親）オブジェクトへの参照を保持しており、プロパティを検索して自分のハッシュ表に見つからない場合にはプロトタイプに検索を委譲する (図 3.13)。これにより、自分からルートまで

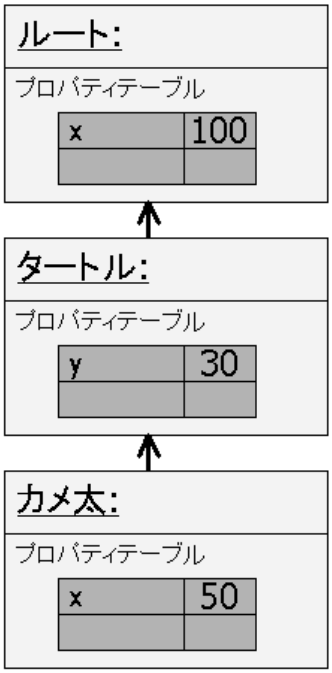


図 3.13: プロパティの参照

のプロトタイプ連鎖のどこかでプロパティが定義されていればそれが参照される。プロパティに値を設定するときはこのような検索は行なわず、直接自分のハッシュ表に登録する。これにより、親側でのプロパティ変更はすべての子供に波及するが、子供側での変更は親や兄弟に影響することはないという性質が実現される。

図 3.14 にメソッド実行の様子を示す。クラス OxObject では invoke() というメソッドを実装しており、メソッド名（文字列）と引数リスト（OxObject の配列）を指定してこのメソッドを呼び出すことで各オブジェクトのメソッドが呼び出せる。invoke() ではまずメソッド名でプロパティを検索し、ブロック値（OxBlock のインスタンス）が取り出せれば、それをドリトルで記述されたメソッドとして実行する（OxBlock のメソッド invoke() を実行する）。見つからなかった場合には、組み込みメソッドの表を検索し、見つかったメソッドをリフレクション機能を通じて直接呼び出す。いずれも見つからない場合はその

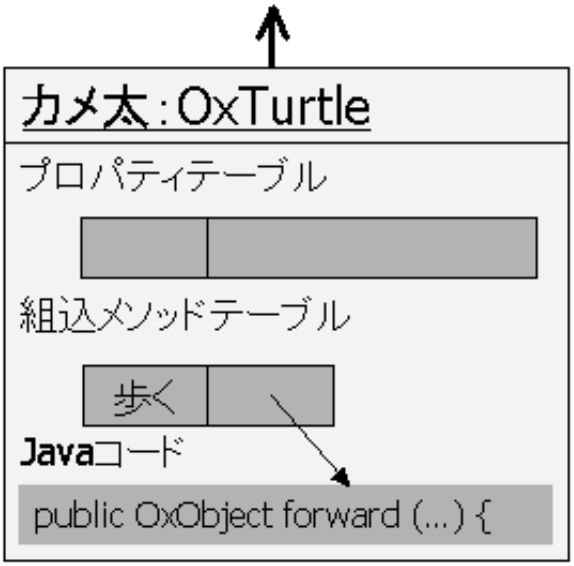


図 3.14: メソッドの検索

呼び出しは失敗し、未定義オブジェクトが返される。

OxBlock はブロック内部のコードの抽象構文木を保持しており、invoke() はインタプリタによりこの抽象構文木を解釈実行する。OxBlock は生成されたときのコンテキストを記憶しており、これを通じて変数を検索するため、ブロックの外側で定義されている変数を参照/更新できる。ただし、ブロックがメソッドとして起動される時はこのコンテキストは使用されず、そのメソッドを保持しているオブジェクトがコンテキストとして使用される（メソッドがそのオブジェクトの変数群を読み書きするため）。

3.4 言語仕様の考察

本節では、ドリトルの設計で採り入れた特徴を中心に、言語仕様を考察する。

3.4.1 クラスを用いないオブジェクト生成

ドリトルでは、クラスを用いずにオブジェクトを生成するプロトタイプ方式のオブジェクト指向を採用した。

従来のオブジェクト指向言語では、プログラミングの前にクラスやインスタンス等のオブ

ジェクト指向の基礎概念を理解する必要があり、習得の困難さが指摘されている [20][39] [42]。

以下に、クラスを用いる方式とクラスを用いない方式 (プロトタイプ方式) を比較する。

クラスによるオブジェクト定義

現代のソフトウェアはコンポーネント技術や GUI の発展により、オブジェクト指向を活用した開発が主流になっている。このような開発で使われる言語はオブジェクト指向を標準でサポートしているが、プログラミングの中で継承を含むクラスの定義が必要であり、コーディングの前にクラス設計を行う必要がある。Kölling et al. [20] は、高等教育における入門用言語において、クラス設計が必要な従来のオブジェクト指向言語の習得が難しいことを報告している。

Smalltalk-80 [12] [16] [44] は Alan Kay らにより教育で使われることを視野に入れて設計されたオブジェクト指向言語である。言語全体がオブジェクト指向で統一され、メッセージ送信によって動くモデルは今回の言語の参考にした。ただし、クラス階層の理解とクラス定義は初心者がプログラムを作る上で負担が大きいという判断から、今回の言語ではクラス方式のオブジェクト指向を採用しなかった。

図 3.15 に Smalltalk-80 のプログラム例を示す。このプログラムでは、既存のタートル

```
class name NewTurtle
superclass Turtle
instance variable names
  step
instance methods
  newstep:  s
    step←s
  run:  n
    self forward:  (n * step)
```

図 3.15: Smalltalk-80 でのクラス定義例

クラスから、新しいクラスを定義している。

クラス方式のオブジェクト指向言語は、Smalltalk-80 のほかに Java [13]、C++ [35]、

Python [22]、Ruby [72] などが存在する。これらの言語を教育用言語として見た場合には、上記の Smalltalk の問題点がそのまま当てはまる。

クラスを用いないオブジェクト生成

Ungar が 1987 年に設計した Self [31] [37] では、Lieberman [21] の提唱した委譲 (delegation) の概念を取り入れることにより、クラスを用いず、プロトタイプオブジェクトの複製によりオブジェクトを生成することが可能である。

プロトタイプ方式のオブジェクト指向 [30] [32] は、インターネット閲覧ブラウザ (Web ブラウザ) 上で動作する JavaScript [6] [38] でも採用されている。

プロトタイプ方式は、クラス方式と比べて次の利点を持つ (図 3.16)。

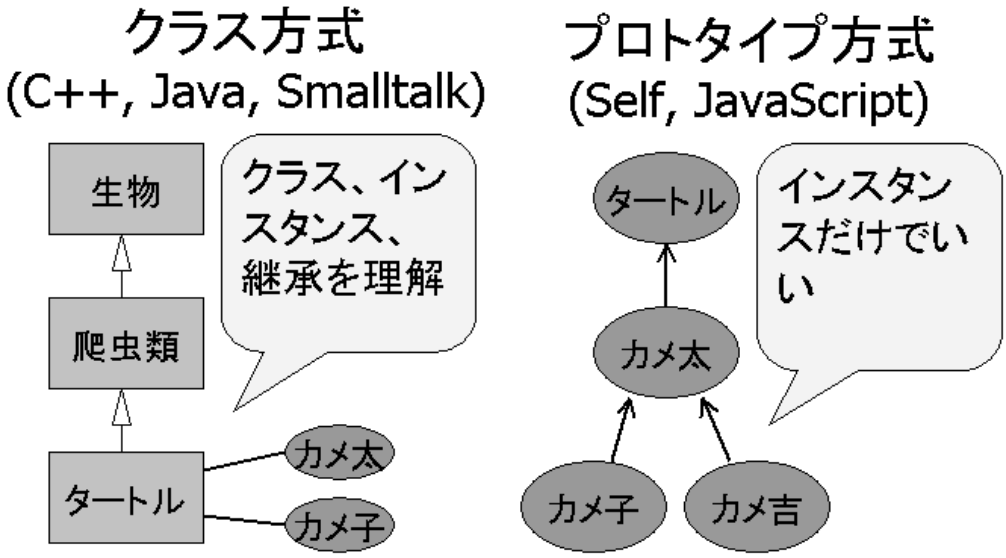


図 3.16: オブジェクト指向方式の比較

- クラス定義が不要である。
- 複雑なクラス階層を理解する必要がない。
- 複製というオブジェクトの生成モデルは単純で理解しやすい。

比較のために、Smalltalk-80 とドリトルのコードを対比する。ここでは「指定した倍率だけ大股で歩く」動作をする「走る（run）」というメソッドを持つ新しいタートルを作

る場面を考える。

Smalltalk-80 のプログラム (図 3.15) では、既存のタートルクラスから、新しいクラスを定義している。実際には、このコードのほかにオブジェクトを生成するコードが必要であり、ひとつのオブジェクトを作るために、クラスを定義してオブジェクトを生成する 10 行以上のコードが必要になる。また、構文中には「class」「superclass」「instance variable」「instance methods」「self」等の多数の予約語が出現するため、学習時にこれらの理解が必要になる。クラスは目に見えない抽象概念であり、さらにそれらの階層関係を理解しなければならないことから、初心者にとってオブジェクトを拡張することは難しい作業になる。

図 3.17 にドリトルのプログラム例を示す。このプログラムでは、タートルオブジェク

```
カメ太＝タートル！作る。
カメ太：歩幅＝「|s|step ＝ (s)」。
カメ太：走る＝「|n|！(n ＊ step) 歩く」。
```

図 3.17: ドリトルでのオブジェクト定義例

ト「カメ太」に 2 つのメソッドを定義している。「歩幅」メソッドでは引数として歩幅 s を受け取り、カメ太のプロパティ「step」に代入する。「走る」メソッドでは引数として歩数 n を受け取り、(n * step) の距離だけ移動する。

プロトタイプ方式の言語においては、プロトタイプオブジェクトを複製してオブジェクトを作り、それに必要なメソッドを定義する。複製というわかりやすい操作でオブジェクトを生成し、実際のオブジェクト生成を含めて数行で記述できるのが特徴である。

クラス方式の言語では、あるクラスから参照できる変数は、プログラム作成時にクラスの継承定義から宣言的に決定される。一方、プロトタイプ方式の言語では、あるオブジェクトから参照できる変数は、その時点でオブジェクトがリンクしている親オブジェクトの状態 (リンク関係や親オブジェクトの変数) によって、実行時に動的に決定される。この性質は、主に大規模なプログラムを書いた場合に、プログラムの動作について見通しを悪くする原因となる可能性がある。しかし、今回行った第 4 章と第 5 章の実験授業で観察した限りでは、生徒の理解に問題になることはなかった。

3.4.2 メッセージ送信のカスケード

ドリトルのプログラムでは、「メッセージ送信で返されるオブジェクトをレシーバにして、続くメッセージを送る」カスケード送信を多用する。この結果、プログラムは一連のメッセージ送信の列が横に並ぶ形になる。

この記法は次の利点がある。

- レシーバを何度も書く必要がないため、プログラムの記述が簡潔になる。
- ひと固まりの仕事をするメッセージ列が一目でわかる。
- 通常の言語に比べて行数が 1/3 程度に短くなるため、画面上でプログラムの広い範囲を見渡せる。

図 3.18 に、三角形を描くプログラムをカスケード送信を使って記述した例と使わずに記述した例を示す。

```
// カスケードを使わない例
カメ太=タートル!作る。
カメ太!100 歩く。
カメ太!120 右回り。
カメ太!100 歩く。
カメ太!閉じる。
三角=カメ太!図形にする。

// カスケードを使った例
カメ太=タートル!作る。
三角=カメ太!100 歩く 120 右回り 100 歩く 閉じる 図形にする。
```

図 3.18: メッセージのカスケード送信

カスケード送信はひとつのオブジェクトに次々とメッセージを送るプログラムを簡潔に記述できるが、「次のメッセージは直前のメッセージが返すオブジェクトに対して送られる」点に注意が必要である。多くのメソッドは実行の結果として自分自身のオブジェクト

を返すが、「図形にする」のように、タートル自身ではなく新たに生成したオブジェクトを返すメソッドが存在する。

カメ太!100 歩く 120 右回り 100 歩く 閉じる 図形にする (青) 塗る。

この例のように「図形にする」の後に「(青) 塗る」を加えた場合には、「歩く」「右回り」「閉じる」「図形にする」はカメ太へのメッセージになるが、「塗る」は「図形にする」で返されたオブジェクトへのメッセージになる。

3.4.3 中置記法と変数の参照

ドリトルでは予約語を定めず、変数を表す記号[6] も用意していない。識別子は、プログラム中に出現する位置に応じて、変数ないしメソッド名と解釈される。「！」記号の左側の識別子はレシーバである。それ以外の識別子が出現するとメソッド名になる。

数式等の中置演算子を丸かっこ内およびブロック内に単独で記述できるようにした。メッセージ式と中置式の区別は構文から判断する。

これらの設計は、次のような利点がある。

- 予約語が存在しないことで、ユーザが「使っていけない言葉」を意識せずプログラミングを行うことができる。
- 変数を表す記号が存在しないことで、キーボードからの記号の入力を減らすことができる。
- 数式や比較式の中で、変数をそのまま書ける。

メッセージ引数に変数を記述する場合は、中置演算子と同様の扱いとして、丸かっこで囲み、メソッド名と区別する。

3.4.4 メソッドの定義

メソッドはブロックをオブジェクトのプロパティに代入することにより定義する。これは次のような利点がある。

[6] Perl など一部の言語では、`$var` のように、名前の前または後に記号を付けることで変数を表す。

- 特別な構文が不要であり、代入やブロックという基本的な構文を組み合わせてメソッドを定義できる。
- メッセージ送信のカスケードと組み合わせることで、多くの場合メソッドを 1 行で記述できる。

3.4.5 日本語によるプログラム

構文に日本語を許すことで、学習者の敷居を下げる効果があると考えられる。

まず、標準オブジェクト名やプロパティ名、メソッド名などの識別子が日本語であること、そして記号に日本語の引用かっこや読点を使えるようにしたことで親しみやすさが増し、英語を未習得の生徒でも使うことが可能になる。

日本における計算機環境では、同じ文字を表すために 2 種類の文字コード (16 ビット文字と 8 ビット文字) が使われることがある。これらは、計算機側の歴史的な都合で区別されているが、計算機学習にとって本質的な問題ではない。また、表示するフォントによっては画面上で区別がつかないことがある。このような学習の本質でない問題でエラーが発生したりその解決で苦労することは、本来の学習内容以外に労力を使い、生徒のやる気を損なう可能性がある。

そこで、日本語での入力につきものの 16 ビット文字（いわゆる全角）と 8 ビット文字（いわゆる半角）の問題についても、できるだけ両者を区別しない（同一の文字として扱う）ようにすることで、頻繁な日本語入力の切り替えを減少するようにした[7]。表 3.5 にドリトルが同等に扱う文字の規則を、表 3.6 にドリトルが同等に扱う記号の例を示す。

表 3.5: 同等に扱う文字の規則

文字種類	規則
英字	大文字と小文字を区別しない。8 ビット文字と 16 ビット文字を区別しない
数字	8 ビット文字と 16 ビット文字を区別しない
仮名	8 ビット文字と 16 ビット文字を区別しない

[7]第 4.3.4 節では、文字や記号の区別について実験授業の評価を踏まえた詳細を扱う。

表 3.6: 同等に扱う記号の例

記号	用途
[〔 「	ブロックの開始
] 〕 」	ブロックの終了
" “ 『	文字列の開始
" ” 』	文字列の終了
. ． 。	文末および小数点

3.5 情報教育に適したプログラミング言語のまとめ

本章では、筆者の設計したプログラミング言語「ドリトル」を説明した。

ドリトルは学校教育用に設計したオブジェクト指向言語である。日常会話で相手に呼びかけるように、画面上のオブジェクトにメッセージを送る形でプログラムを記述する。従来のプログラミング言語では、JavaやC++が持つオブジェクト指向の記述能力とBASICやLOGOが持つ入門時の学びやすさを両立させることは難しかった。ドリトルは簡潔な構文とオブジェクト指向の組み合わせにより、学びやすさと記述力を両立した。

ドリトルはオブジェクトにメッセージを送る単純なモデルで動作する。文法は記号が少ない構文とし、日本語の命令語を使い記述する。階層的な構造を取らずにプログラムを記述できることで、学習者の負担を軽くする工夫を行った。プログラムは1行ずつが単独で意味を持つ形とし、学習時には1行ずつ追加して試しながら学べるようにした。記述能力としては、数値や文字列などの基本的なデータ型を繰り返しや条件分岐を使い操作することで、学校教育で学ぶ基本的なアルゴリズムを記述できる。クラスを用いないプロトタイプ方式によるオブジェクト指向とGUI部品等のオブジェクトを組み合わせることで、従来の入門用言語では難しかった実用的なプログラムの記述が可能になった。外部機器やネットワークとの通信を可能にしたことで、制御や通信の学習が可能になった。

ドリトルの実装はJava2を用いて行った。Javaで記述することにより、ドリトルはWindows, Macintosh, Linux など、学校教育で使われる、Javaが稼動するさまざまなプラットフォームで動作させることが可能である。

第4章　情報教育とプログラミング

本章では、小規模な2種類の実験授業により、設計したドリトルの言語仕様が妥当であることを確認する。ドリトルは小中学校からの授業を想定して設計されているが、最初の実験では論理的な思考に比較的慣れていると考えられる高校生を対象に授業を行い、学習の様子を詳細に観察した。続いて、学校教員に対する授業を行い、学校教育で利用する際に大きな問題がないことを確認した。

4.1　高校での実験授業

4.1.1　実験概要

調査対象

実験授業は筑波大学附属高校で実施した。受講した生徒は1年生3名である。以下に生徒のプロフィールを示す。

1人はVisualBasicによるプログラミング経験があった。他の2人はWWWブラウザやワードプロセッサ等の使用経験があったものの、プログラミング経験はなかった。また、このうち1人はプログラムがどんなものであるかという知識を持っていたが、もう1人はプログラミングやソフトウェアの概念をまったく知らなかった。

調査・実験方法

ドリトルを使ったプログラミングの授業を3回実施した。時間は放課後の1時間前後を使い、隔週程度の間隔で行った。講師は筆者が担当した。

1. 観察とアンケートによる評価
   授業中の生徒の発言とアンケートの記録から、生徒が学習する際に獲得した知識や

理解しづらかった個所などを観察した。

2. 作成されたプログラムの分析

   3 人の生徒はプログラミングの知識や経験に多少の差があったため、毎回の授業では、前半に言語の概念を説明した後、後半に個々のレベルに合った目標を設定してプログラミング実習を行った。作成されたプログラムを分析することで、生徒が理解している概念を調べた。

分析方法

学習の観察については、プログラミングの概念を持たなかった生徒の発言を中心にたどることで、プログラミングの体験を通して、どのような知識を獲得するかを分析した。

プログラムの分析については、生徒の作品プログラムで使われている命令やアルゴリズム等を集計することで、どのようなプログラミングの概念を理解しているかを分析した。

4.1.2 授業の進め方

実験授業は放課後を利用し、1 回あたり 1 時間の授業を 3 回実施した。表 4.1 にカリキュラムを示す。

表 4.1: 実験授業のカリキュラム

授業	内容
1	メッセージ、オブジェクトの説明 タートルグラフィックスを用いた対話的な実行
2	タイマーによる一定間隔の繰り返し実行 複数のオブジェクトを操作
3	メソッドの定義 オブジェクトの継承

使用した環境として、1 人 1 台のノートコンピュータを用いた。OS は Windows98、画面の解像度は 800 × 600 である。ネットワークには特に接続しなかった。

授業では、前半の 30 分で 10 行程度のサンプルプログラムを解説し、そのプログラムを入力して動作を確認する形で学習を進めた。後半の 30 分では、課題のプログラムを作成してもらった。授業は、プログラム作成中の生徒の発言や、後で行ったアンケート、作成されたプログラムから、生徒が獲得した知識を評価した。授業は次のように進めた。

1 回目の授業では、新しいオブジェクトを生成する方法、複製する方法、オブジェクトを操作する方法を学んだ。図 4.1 に、1 回目の授業で解説したプログラムを示す。このプログラムでは、タートルグラフィックスを用いて三角形を描く方法を示している。授業では、このプログラムを参考にして、自由な図形をドリトルで描画する課題を出し、生徒に挑戦してもらった。

```
カメ太 = タートル！作る。
カメ太！ペンなし。
カメ太！200 歩く。
カメ太！120 右回り。
カメ太！ペンあり。
カメ太！100 歩く。
カメ太！120 右回り。
カメ太！100 歩く。
カメ太！閉じる。
三角 = カメ太！図形にする。
三角！(青) 塗る。
三角！0 200 移動する。
```

図 4.1: 1 回目のサンプルプログラム

2 回目の授業では、タイマーオブジェクトを使い、繰り返し作業するプログラムを学んだ。図 4.2 に、2 回目の授業で解説したプログラムを示す。このプログラムでは、タイマーを用いて 2 匹のタートルと三角形を移動するアニメーションの作り方が示されている。授業では、このプログラムを参考にして、自由な図形のアニメーションを作ることを課題とした。

3 回目の授業では、オブジェクトにメソッドを教える方法と、オブジェクトの性質を他

```
カメ太 = タートル！作る。
カメ子 = タートル！作る。
時計 = タイマー！作る。
カメ太！100 歩く 120 右回り 100 歩く。
三角形 = カメ太 ！ 閉じる 図形にする。
三角形！0 150 移動する。
時計！1 間隔 10 時間。
時計！「三角形！36 右回り。カメ太！15 歩く。カメ子！36 右回り」実行。
```

図 4.2: 2回目のサンプルプログラム

のオブジェクトに継承する方法を学んだ。図4.3に、3回目の授業で解説したプログラムを示す。このプログラムでは、オブジェクトに命令を定義し、それを呼び出して実行するプログラムの作り方が示されている。授業では、3回の授業で学んだ知識を使い、自由課題でプログラムを作成した。

```
カメ太 = タートル！作る。
時計 = タイマー！作る 1 間隔 10 時間。
三角形 = カメ太 ！ 100 歩く 120 右回り 100 歩く 閉じる 図形にする。
三角形！(赤) 塗る。
三角形：ぐるぐる = 「時計！「三角形！ 36 右回り」実行」。
三角形！ぐるぐる。
```

図 4.3: 3回目のサンプルプログラム

4.1.3 授業結果

実験授業の結果を示す。最初に、生徒の観察とアンケートによる評価を示す。続いて、生徒が作成したプログラムを分析した結果を示す。

観察とアンケートによる評価

アンケートと授業中の生徒の発言から授業を評価した。表 4.2 に、アンケートに書かれた生徒の感想を示す。アンケートから、生徒が計算機の基本原理を理解し、プログラミングの楽しさを体験することができたことがわかる。

表 4.2: 生徒の感想

- プログラムは上から順に実行されることがわかった
- 同じ動作を 2 回記述しないで済ませるのは便利だと感じた
- 自分で考えたことが、うまく画面に現れるととても嬉しかった
- どんなことをしようか決めてからプログラムを組むのはパズルのようで面白かった

(1) 計算機の基本原理を理解できた

3 人の生徒のうち、初めてプログラミングを体験した生徒が授業中に行った発言から、プログラミングを行いながら次の概念を学んだものと判断した。

- プログラミングの実行の進み方に一定の規則があることを理解する
  命令を順に処理する逐次実行は、最も基本的な計算機の動作である。「プログラムは上から実行されることがわかった」という感想から、生徒はどのようなプログラムでも、基本的に上から下に順に実行されていくという原理を発見できた。

- 画面の仕組みを理解する
  計算機の画面として、一般にビットマップディスプレイが使われている。ビットマップディスプレイでは、文字やグラフィックスなど画面に表示されるすべてのものは、画面上の点 (画素) の並びとして表現される。「なぜ 5cm 歩くのに 100 歩く必要があるのだろう。ひょっとして、画面の 100 個の点だけ進むのかな」という授業中の発言から、生徒は計算機の画面が cm や mm といった単位で取り扱われるものではなく、点の集まり (画素) として取り扱われていることに気付くことができた。

- 計算機がプログラムで動作していることの発見
  計算機では、周辺機器の制御やユーザとの対話的なインターフェースの機能を提供するために、オペレーティングシステムが動作している。今回の実験授業で使用した計算機の中では、Windows である。「カメさんを動かすのにプログラムが必要なら、このパソコンのマウスカーソルを動かすために別のプログラムが動いているのですか? もしウィンドウを表示するプログラムも入っているなら、ずいぶん大きなプログラムなんですね」という授業中の発言から、生徒はタートルを動かすためにプログラムが必要なことからの類推として、マウスを動かしたときに画面でカーソルが動くのは、計算機の中にプログラム (オペレーティングシステム) が内蔵されているためであると気付くことができた。また、カーソル以外にもウィンドウの表示やアプリケーションプログラムの動作などがすべてプログラムで行われていることから、膨大なソフトウェアで計算機が構成されていることを類推することができた。

計算機の原理を、説明を聞くだけで理解することは容易ではない。今回の実験授業では、プログラミングという自分たちの能動的な体験と結び付けて学習することで、逐次実行の原理、画素の存在、オペレーティングシステムの発見など、計算機のさまざまな側面を生徒が発見し、それを補う形での授業を行うことができた。

(2) プログラミングの楽しさを体験できた

3回の授業を通して、楽しい雰囲気の中でプログラミングを体験することができた。表 4.2 に、最終回の授業で行ったアンケートに書かれた生徒の感想を示す。自分の作品が動く喜びや、それを考えるプロセスの楽しさを感じていたことがわかる。また、生徒から「授業が終ってからも自宅で使ってみたい」という要望があったため、最終回の授業でドリトルの実行プログラムを配付し、インストール方法を説明した。

作成したプログラムの分析

生徒が作成したプログラムで使われているプログラミング上の概念を分析することにより、生徒の理解度を評価した。

(1) 言語を理解し活用することができた

ドリトルのプログラムは簡潔であり、実行結果が視覚化されているため、数行の例題プログラムからプログラムと結果の動作の対応関係を理解できた。特に、タイマーオブジェクトの導入により、図形の作成、時間の操作方法の定義、動作のアルゴリズムの構築という複合した作業の成果としてアニメーションを表現するプログラムを作成することができた。その結果、図 4.4、図 4.5 に示す 3 人の生徒の共同作品の中で、表現したい図形を描き、色を変えながらアニメーションとしてダイナミックに動くプログラム作品を作ることができた。このプログラムを実行すると、2 個のタートルオブジェクトが左右に対称に動きながら半分ずつの赤いハートを描き、図形オブジェクトを生成する。左右の図形オブジェクトは、あたかも 1 個のオブジェクトのように同時に上昇し、突然青い色に変化して、左右に分かれて落下する。このプログラムでは、動作の繰り返し、メッセージのカスケード実行、タイマーオブジェクトによる定期的な繰り返し実行、複数オブジェクトの同時実行、タイマーオブジェクトの実行終了の待ち合わせなど、3 回の授業で学んだ概念が正しく使われていることがわかる。

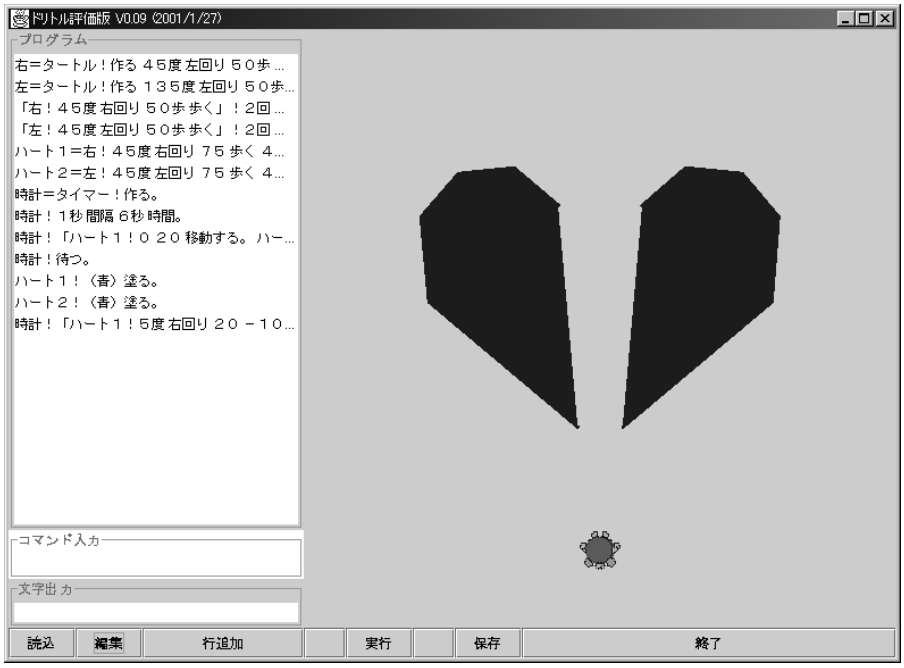


図 4.4: 高校生の作品例（1）

```
右＝タートル！作る 45度 左回り 50歩 歩く。
左＝タートル！作る 135度 左回り 50歩 歩く。
「右！45度 右回り 50歩 歩く」! 2回 繰り返す。
「左！45度 左回り 50歩 歩く」! 2回 繰り返す。
ハート１＝右！45 右回り 75 歩く 45 右回り 171 歩く 図形にする（赤）塗る。
ハート２＝左！45 左回り 75 歩く 45 左回り 171 歩く 図形にする（赤）塗る。
時計＝タイマー！作る。
時計！1秒 間隔 6秒 時間。
時計！「ハート１！0 20 移動する。 ハート２！0 20 移動する」実行。
時計！待つ。
ハート１！（青）塗る。
ハート２！（青）塗る。
時計！「ハート１！5度 右回り 20 -10 移動する。
  ハート２！5度 左回り -20 -10 移動する」実行。
```

図 4.5: 高校生の作品プログラム（1）

(2) 現代的なプログラミングを体験できた

既存のオブジェクトに「作る」というメッセージを送り複製することで、複製されたオブジェクトが元のオブジェクトの性質を引き継いでいることを学んだ。また、新しいオブジェクトを追加したり、オブジェクトにメソッドを定義することで、使える命令を増やしていけることを体験した。その結果、プログラミング経験のあった生徒は図 4.6、図 4.7 に示すプログラムの中で、1 個のオブジェクトにメソッドを定義し、他のオブジェクトに継承する作品を作ることができた。このプログラムを実行すると、タートルが落ち葉に見立てた三色の平行四辺形を描き、色ごとに異なる落ち方をメソッドとして定義する。続いてそれぞれの平行四辺形を複製して複数枚の落ち葉を作り、タイマーで順に落下させる。このプログラムでは、オブジェクトの複製によるメソッドの継承をうまく利用することで、複数枚の落ち葉の定義を簡潔に定義する工夫が行われていることがわかる。

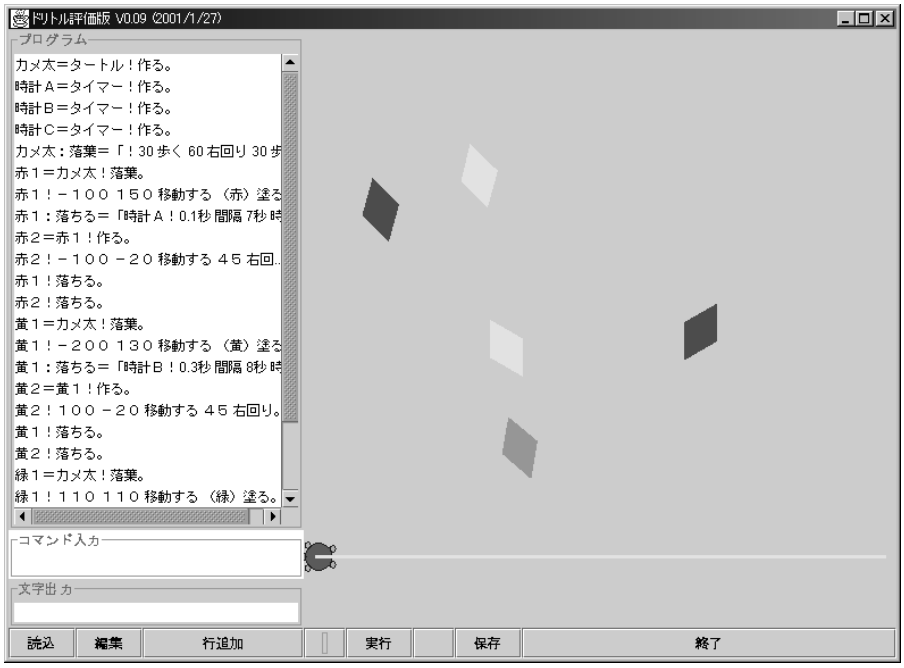


図 4.6: 高校生の作品例（2）

```
カメ太＝タートル！作る。
時計Ａ＝タイマー！作る。
時計Ｂ＝タイマー！作る。
時計Ｃ＝タイマー！作る。
カメ太：落葉
　＝「！30 歩く 60 右回り 30 歩く 120 右回り 30 歩く 閉じる 図形にする」。
赤１＝カメ太！落葉。
赤１！-100 150 移動する （赤）塗る。
赤１：落ちる＝「時計Ａ！「 ！18 右回り 5 -5 移動する」実行」。
赤２＝赤１！作る。
赤２！-100 -20 移動する 45 右回り。
時計Ａ！0.1秒 間隔 7秒 時間。
赤１！落ちる。
赤２！落ちる。
黄１＝カメ太！落葉。
黄１！-200 130 移動する （黄）塗る。
黄１：落ちる＝「時計Ｂ！「 ！30 右回り 10 -13 移動する」実行」。
黄２＝黄１！作る。
黄２！100 -20 移動する 45 右回り。
時計Ｂ！0.3秒 間隔 8秒 時間。
黄１！落ちる。
黄２！落ちる。
緑１＝カメ太！落葉。
緑１！110 110 移動する （緑）塗る。
緑１：落ちる＝「時計Ｃ！「 ！40 右回り -10 -10 移動する」実行」。
時計Ｃ！0.2秒 間隔 6秒 時間。
緑１！落ちる。
カメ太！ペンなし 90 左回り 200 歩く 90 右回り -250 歩く。
カメ太！ペンあり 500 歩く 図形にする （黄）塗る。
```

図 4.7: 高校生の作品プログラム（2）

4.2 教員による評価

教える側の立場である学校教員に意見を聞くために、ドリトルを体験する講習会を実施した。

4.2.1 調査概要

調査対象

講習を受講したのは、兵庫県教育工学研究会[1] に所属する、小学校から大学までの教員 20 名である。参加者の内訳を表 4.3 に示す。参加者のうち、15 名は LOGO、BASIC、PASCAL 等のプログラミングを経験したことがあり、5 名はプログラミングの経験がなかった。

表 4.3: 教員の内訳

学校種別	経験者	未経験者
小学校	3	3
中学校	5	2
高校	2	0
大学	2	0
養護学校など	3	0
計	15	5

調査方法

講習では、言語の概念と文法を 30 分間説明し、続けて 60 分間の実習を行った。講師は筆者が担当した。テキストには高校の実験授業で使った 3 時間分の資料を配布した。受講者の多くは、テキストの例題を入力して試しながら、独自の作品として絵やアニメーションを作成した。

[1] 教育工学に関する話題を扱う研究会。主に兵庫県内の大学教員と学校教員で構成されている。

調査は、観察とヒアリングにより行った。観察は、講習中にプログラムを作成する様子を観察し、質問された内容を記録した。ヒアリングは、講習後に使ってみての感想と授業で使えるかどうかの感想を 1 人ずつ述べてもらう形で行った。

分析方法

設計したドリトルの言語仕様に問題がないことを確認するという観点から、観察とヒアリングの中で、否定的な意見や感想の有無をチェックした。

4.2.2 講習の結果

表 4.4 に参加者の感想を示す。

表 4.4: 教員の感想

- 自分でプログラムを作れるのが面白かった
- 難しいと思っていたプログラムを自分が作れることがわかり、面白かった
- 日本語でプログラムを簡潔に記述できるのでよい
- 従来の言語に比べてプログラムを簡潔に記述できる
- 初心者の自分が 1 時間で使えるようになったのだから、小学生でも簡単に使えると思う。ぜひ教えたい

全体に共通した意見としては、「授業で使いたい。教室で問題なく教えられると思う」という感想が多かった。

プログラミング経験の有無で見ると、今回の講習で初めてプログラミングを体験した教員 5 名からは、「よく理解できた。初めての教員でも容易に習得できる」という感想を得た。プログラミング経験のある教員 15 名からも同様に、「わかりやすい」という感想が多かった。

ただし、講習を観察した結果では、他の言語を普段使い込んでいる 1,2 名の教員は、最初は慣れ親しんだ言語の文法でプログラムを書いてしまう傾向が見られた。新しい言語

への移行は一般に発想の切り替えに時間がかかるが、BASIC や LOGO に慣れた教員の場合、30 分程度でドリトルに慣れることが観察された。

以上の結果から、ドリトルはプログラミングの未経験者を含め、教える側の教員に容易に習得できることを確認できた。

4.3 言語仕様の考察

実施した実験授業から、ドリトルの特徴的な言語仕様を考察する。

4.3.1 タートルグラフィックスの拡張

描いた線を図形オブジェクトにして操作するタートルグラフィックスの拡張は、作品の表現力の幅を広げ、プログラミングの学習にも効果があった。

LOGO に代表される従来のタートルグラフィックスでは、タートルが移動した軌跡を画面上の点として残していた。描いた軌跡は画面上の静止画像として観賞することになる。そのため、三角形を描いた後で六角形を描いたり星を描くといった複雑な図形を描く授業は行えるが、それ以上に授業を発展させることが難しいという問題があった [3]。

ドリトルでは軌跡から図形オブジェクトを生成し、命令を送って操作できるようにした。その結果、描いた軌跡を静止画として観賞するだけでなく、メッセージを送り移動・回転などの操作を行えるようになり、作品の表現力が増した。高校生を対象とした実験授業では、ハートや落葉の形を描いた後で、それをアニメーションとして動かした。自分の描いた形に動きを付けるという意味で、タイマーオブジェクトを学習する動機付けにつながった。

4.3.2 クラスを用いないオブジェクト生成

オブジェクトを複製によって生成するプロトタイプ方式の採用は、クラス定義が不要であることから、オブジェクト指向に対する初期の敷居を下げることができた。

今回行った実験授業では、高校生が 3 時間という限られた時間の中でオブジェクトの概念を扱った。たとえば、図 3.17 に相当するオブジェクト定義は、第 4.1 節で扱った図 4.7

の高校生の作品の中で有効に活用されている。

```
赤１：落ちる＝「時計Ａ！「 ！18 右回り 5 -5 移動する」実行」。
赤２＝赤１！作る。
```

この 2 行では、オブジェクト「赤 1」に「落ちる」というメソッドを定義した後、「赤 1」を複製することでオブジェクト「赤 2」を生成し、「赤 1」に定義したメソッドを「赤 2」に継承している。

このような簡潔な記述は、プロトタイプ方式を採用したために可能になった。仮にクラス方式を採用した場合は、図 3.15 に相当するクラス定義を記述する必要があり、結果として 10 行以上に渡るプログラムの記述が必要になってしまう。

4.3.3 タイマーと並行プログラミング

タイマーオブジェクトを用意したことで、描いた図形をアニメーションで動かすことが可能になった。その結果、プログラムの表現力が増し、生徒の動機付けに役立った。

タイマーオブジェクトを利用することで、定期的な繰り返し実行が行えるため、図形などを間欠的に移動・回転させるアニメーションが可能になる。また、複数のオブジェクトを同時に動作させることが可能なため、並行プログラミングを体験することが可能になった。

タイマーは非同期に実行される。当初はタイマーの終了を待ち合わせる機能を用意していなかったが、高等学校での 2 回目の実験授業の中で、タイマーの実行を待ってからプログラムの実行を次に進めたいという場面があったため、タイマーに「待つ」というメソッドを実装した。

```
時計！ ハート１！ 0 20 移動する。ハート２！ 0 20 移動する」実行。
時計！ 待つ。
ハート１！ (青) 塗る。ハート２！ (青) 塗る。
```

この例は、高校生の作品「ハート」(図 4.5) からの抜粋である。2 行目の「待つ」により、1 行目のタイマーの処理 (赤いハート型の図形の移動処理) が終るのを待ってから、3 行目の処理 (ハート型の図形を青く塗る処理) を実行している。

4.3.4 日本語によるプログラム

構文に日本語を許すことは、学習に対する敷居を低くする効果があった。

高校生のキー入力を観察すると、日本語入力変換の切替が負担になっていることがわかった。命令語などを 16 ビット文字 (いわゆる全角) の日本語で入力した後、8 ビット文字 (いわゆる半角) への切替をせずに、空白文字や記号、英字、数字などを入力する学習者が多い。また、表示するフォントによっては画面上で区別がつかないことがある。

16 ビット文字と 8 ビット文字については、開発当初は表 3.6 の記号のみを区別しない仕様としていた。しかし、高等学校での 1 回目の実験授業の中で、生徒たちが数字を 16 ビット文字で記述することによるエラーが多く観察されたため、8 ビット文字と 16 ビット文字の数字を区別しないで記述できる仕様に修正した[2]。

これらの改良により、プログラミング時に日本語入力を切り替える頻度を減らすことができた。また、8 ビット文字と 16 ビット文字を区別しない仕様にしたことで、文字入力の些細な揺らぎによる不要なつまづきを減少させる効果があったと考えている。

4.4 情報教育とプログラミングのまとめ

設計した言語が教育現場で使えることを確認する目的から、教員と少人数の生徒を対象に実験授業を実施した。その結果、ドリトルの設計方針に大きな問題がないことを確認した。また、授業の観察から言語仕様や処理系の実装の問題点を考察し、必要な改良を行った。得られた成果は以下のとおりである。

[2]その後の授業においても、次のような問題点が報告されたことから、授業を担当した教員たちと相談しながら、他の文字の扱いについても仕様を修正した。

- 三重県松阪市立鎌田中学校の授業 (第 5 章) において、8 ビット文字の「。」を入力したために構文エラーになったが、16 ビット文字と見かけ上の区別がつかなかったため、数日間エラーの原因を発見できないという現象が発生した。そのため、8 ビットの日本語文字 (いわゆる半角かな) についても「8 ビット文字と 16 ビット文字を区別しない」という改良を行った。
- 静岡県藤枝市立西益津中学校で行われた授業 [41] において、変数名に 8 ビット文字の「x」と 16 ビット文字の「X」を混在して使ったためにプログラムが動かず、エラーの発見も難しいという現象が発生した。そのため、英字の名前については「8 ビット文字と 16 ビット文字を区別しない」「大文字と小文字を区別しない」という改良を行った。

- 入門用のカリキュラムとテキストを作成した。導入時に使用できる 3 時間分の授業内容を検討し、テキストを作成した。これらは高校生を想定して作られたものであり、中学生を対象とするときは 6 時間程度の授業を行える内容になっている。

- 高等学校で 3 人の生徒に授業を行った。生徒たちはドリトルの概念や文法を理解し、いくつかの作品プログラムを作成した。この授業を通して、ドリトルの言語仕様に大きな問題がないことを確認した。

- ドリトルのプログラミングを通して、プログラミングを初めて体験した生徒が計算機の原理を発見していくことができた。この観察により、プログラミングの学習を通して計算機の動作原理を学習できるという仮説を検証できた。

- 小学校から大学までの教員を対象にドリトルの講習を実施した。その結果、ドリトルは教える立場の教員に対しても特に問題なく受け入れられ、1 時間程度の講習で教員が基本的なプログラミングを行えるようになることを確認した。実際に授業を行うためには、さらに教員側の準備と学習が必要である。教員が言語の仕様を理解し、授業を行うために、ドリトルのマニュアルや実験授業のテキストなど、各種資料の提供を行う必要がある。

- LOGO や BASIC など、他の言語を使っていた教員もドリトルを使うことができた。ただし、構文の違いやオブジェクトの概念に対する慣れが必要なことから、ドリトルを違和感なく使えるようになるまでに 30 分程度を要することがあった。すでに他の言語を用いてプログラミングを行っている教員のために、マニュアルで言語の特徴を説明する必要がある。

- 生徒と教員が学習する様子の観察から、日本語文字の扱いやタイマーの待ち合わせなど、いくつかの仕様を見直し、処理系に反映することができた。

本章では、教員と少人数の生徒に対する実験授業を扱った。次章では、より大規模な授業による評価について述べる。

第5章　学校教育での実践と評価

本章では、教育用に設計したドリトルを中学校の授業で使用する実験を行い、実際の教育現場で使用した場合の効果や課題を評価する。受講した生徒数は 132 人である。評価は、数種類のアンケート、筆記試験、作品分析により行った。

5.1　実験概要

5.1.1　調査対象

実験授業は公立中学校 (三重県松坂市立鎌田中学校) で実施した。期間は 2001 年 10 月から 2002 年 3 月で、技術科の授業として 11 回の授業が行われた。

対象となった生徒は第 2 学年の生徒 132 人である。学年あたりのクラス数は 4 クラスであり、1 クラスあたり平均 33 人で授業を行った。生徒たちは前学期に WWW 閲覧など計算機の簡単な操作を習ったが、プログラミングの授業は初めてであった。

担当した教員は技術科の担当教員である。過去に LOGO 言語の授業を行った経験があるが、ドリトルでの授業経験はない。

筆者らは、担当教員に事前の講習を行い、オブジェクト指向の考え方や、ドリトルの使い方についての情報を伝えた。また、ドリトルのマニュアルや過去の講習テキストを提供し、これらの資料を元に教員が授業用のテキストを作成した。教員から出された進め方の疑問点などに対しては、随時電子メールで連絡を取りながらサポートを行った。学期末テストの問題とアンケートは担当教員と共同で作成した。

5.1.2　調査・実験方法

以下の 4 種類の方法により、実験授業を評価した。

1. 授業ごとのアンケートによる評価
   生徒の興味や達成感の変化を調べるために、毎回の授業でアンケートを実施した。アンケートは「楽しさ」「達成度」「難しさ」の項目があり、それぞれについて「そう思う」から「思わない」までの 4 段階から選択する形とした。

2. 2 回の定期試験による評価
   生徒の理解度を定量的に調べるために、2 学期末と 3 学期末の定期試験にドリトルの問題を出題し、生徒の回答を分析した。試験は技術・家庭科の技術科の部分（25 分間、50 点満点）として行なわれた。家庭科と合わせて実施したため、テキストやノートは持ち込まない形とした。

3. 授業実施後の理解度アンケートによる評価
   生徒が理解できたと感じる度合いを調べるために、最終回の授業でプログラミングの概念についてのアンケートを実施した。アンケートは 8 個の質問項目があり、それぞれについて「そう思う」から「思わない」までと「意味がわからない (不明)」という 6 段階から選択する形とした。

4. 作成されたプログラムの分析
   プログラミングの概念が実際に活用される度合いを調べるために、生徒作品を分析した。生徒たちは、授業で学んだことを活用して自分の作品プログラムを作った。作品は生徒ごとのフロッピーディスクに保存されており、それらを回収して分析を行った。

アンケートの用紙と定期試験の問題は、事前に教員に送付した。結果の用紙とプログラムファイルは、教員から複製を提出してもらい分析を行った。

5.1.3 分析方法

得られた評価データに対して、以下の方法で分析を行った。

1. 授業ごとのアンケートによる評価
   授業が進むにつれて生徒の学習がどのように変化したかを評価するために、11 回の

授業を「2 学期に実施した前半 5 回」と「3 学期に実施した後半 6 回」に分けることで、符号検定 (Sign Test) による比較を行った。

符合検定は次の手順で行った。まず、有効でないデータを除くために、前後半のいずれかで 3 回以上欠席した生徒を対象から除いた。この結果、集計対象は 110 人になった。次に、各項目の値を 4 段階の数値に置き換え、各生徒の前半 5 回と後半 6 回の平均値を比較した。続いて、符合検定の定義により、平均値が同じもの (表 5.2 の 0 の人数) を対象からはずし、後半の値が前半より大きい人数 (表の + の人数) と小さい人数 (表の − の人数) から検定を行った。検定の結果により、評価項目の値が有意に変化したかどうかを評価する。

2. 2 回の定期試験による評価
   プログラミング概念の理解度をより細かく分析する目的から、問題ごとにチェックポイントを設定し、回答の内容を検討することで、出題時に意図した概念を生徒が理解しているかどうかを判定した（概念理解率）[1]。

   概念理解率では、採点上は不正解とされた場合でも、次の条件に当てはまる場合は「正解ではないが、概念を理解している」と判定している。

   - プログラムの概念は理解しているが、図形の数学的な知識など、他の概念の誤りで誤答となった場合
   - プログラムの概念は理解しているが、記号の抜けや書き間違いなど細かなミスにより構文が不完全であった場合

3. 授業実施後の理解度アンケートによる評価
   5 段階の回答のうち、肯定度の高い上位 2 つを「肯定的な回答」とし、それ以外との比率を比較した。

4. 作成されたプログラムの分析
   生徒が作ったプログラム作品から、使われている機能を取り出し集計した。集計は

[1]試験の結果は、担当教員により成績評価のための採点も行われた。採点では理解不足による誤りだけでなく、単純なミスなども誤りとして扱われてしまうため、今回は成績評価のための採点をそのまま評価対象として扱うことはしなかった。ただし、授業結果 (第 5.2.2 節) では、参考のために成績評価のための採点結果 (各定期試験の平均点) を含めて報告する。

生徒ごとに行った。ひとりの生徒が複数の作品を作成した場合には、作品のどれかで使われていればその機能を活用しているものとして扱った。

5.1.4 授業の進め方

授業は、担当教員が黒板でその日のポイントを解説してから、生徒が課題を実習する形式で行われた。中学校における 1 時限は 50 分である。授業開始時に計算機の起動に 5 分、授業終了時にアンケートの記入と回収で 5 分が使われるため、毎回の実質的な授業時間は 40 分前後であった。

表 5.1 に、実施したカリキュラムを示す。内容に書かれた毎時間のタイトルは、その時間の授業で使用した資料のタイトルを示す。

表 5.1: 授業カリキュラム

学期	時間	内容
2		《タートルグラフィックス》
	1	使ってみよう
	2	三角形、四角形
		《図形オブジェクトの生成と操作》
	3	星・円・塗る
	4	図形を移動する
	5	いろいろな図形を描く
3		《タイマーによるアニメーション》
	6	タイマー
	7	作品にタイマーを入れる
	8	作品作り
		《GUI 部品（ボタン）の活用》
	9	ボタン
	10	ボタン
	11	作品作り

1,2 時間目は手続き的なプログラミングの基礎概念を理解する授業である。生徒はタートルグラフィックスを使い画面に三角形や四角形などの図形を描くことでプログラミングを体験した。入門段階でタートルグラフィックスを用いる利点は、プログラムが正しく動作しているかを本人が視覚的に判断できるため、正しい動作に至るまで、自分で実行結果をフィードバックしてデバッグ作業を行えることである。

3〜5 時間目はオブジェクト指向の基礎概念を理解する授業である。生徒はタートルオブジェクトにメッセージ “図形にする” を送り、タートルが移動により描いた線を図形オブジェクトにすることで、オブジェクトの生成を学んだ。次に、画面上の図形オブジェクトに色を塗るメッセージを送ることで、「特定のオブジェクトにメッセージを送る」というモデルを学んだ。

5 時間目までの内容で、2 学期の期末テストを行った。

6〜8 時間目はタイマーを理解する授業である。生徒は画面上の図形オブジェクトを一定の速度で移動させるアニメーションを通して、タイマーオブジェクトを体験した。

9〜11 時間目は GUI 部品を体験する授業である。今回は代表的な GUI 部品としてボタンを扱った。生徒はボタンが押されたときの動作をボタンオブジェクトに定義することを通して、メソッドの利用とイベントを処理するプログラミングを体験した。11 時間目の最後には、プログラミング概念の理解度を調査するアンケートを行った。

11 時間目までの内容で、3 学期の期末テストを行った。

5.2 授業結果

5.2.1 授業ごとのアンケート

図 5.2〜図 5.3 に集計結果を、表 5.2 に解析結果を示す。

解析は符合検定で行った。11 回の授業を前半 (2 学期の内容。1〜5 時限) と後半 (3 学期の内容。6〜11 時限) に分け、授業が進むにつれてどのように変化するかを見た。その結果、後半の授業では前半に比較して難しさが増しているが、課題の達成度に大きな変化はなく、楽しさが増しているという結果になった。

集計結果と検定結果から、次のことが読み取れる。

- 無理なく授業が成立した

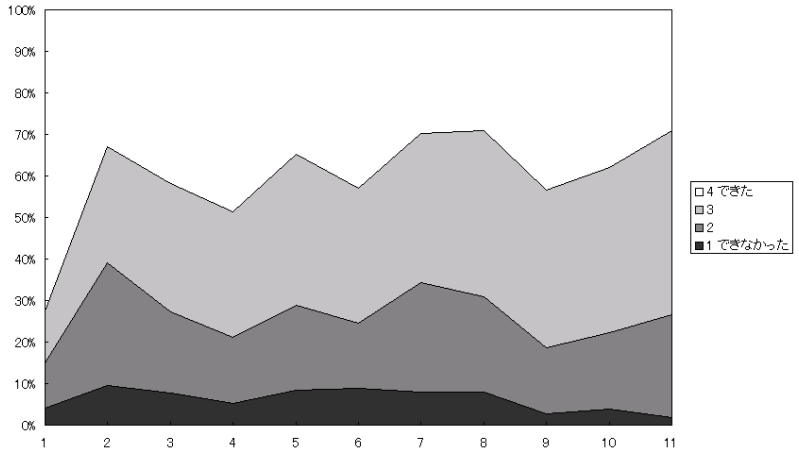


図 5.1: アンケート結果 (達成度)

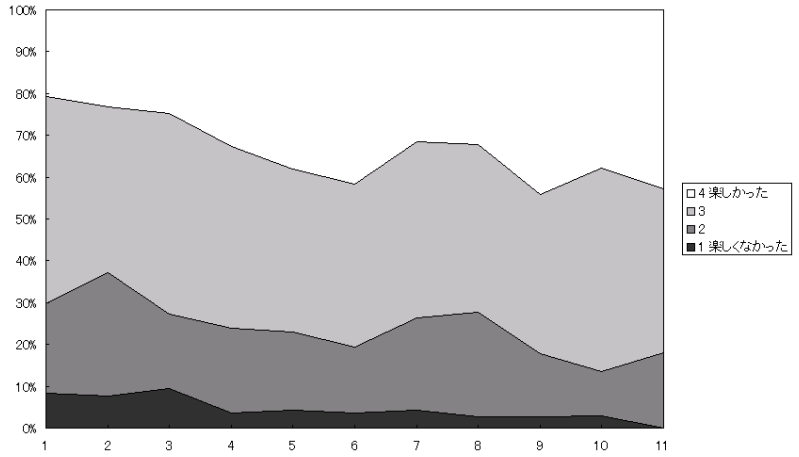


図 5.2: アンケート結果 (楽しさ)

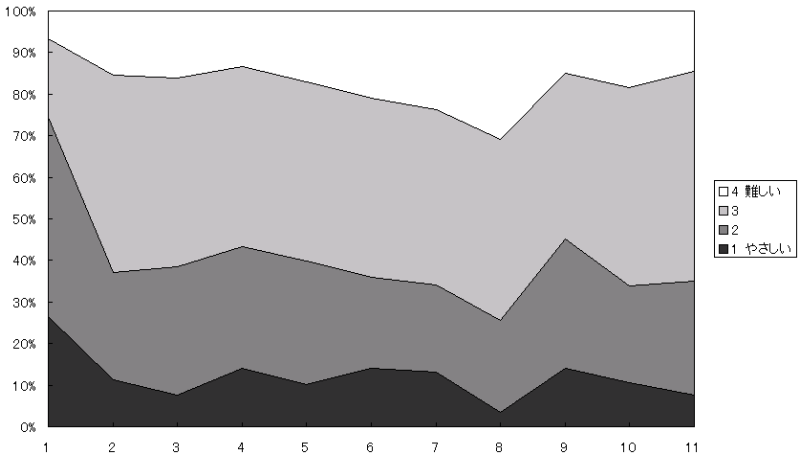


図 5.3: アンケート結果 (難しさ)

表 5.2: 符合検定によるアンケートの解析

項目	変化した人数			検定結果
	＋	−	0	
達成度	40	60	10	達成度は有意な変化なし
楽しさ	71	33	6	楽しさが増加した (1%の有意水準)
難しさ	65	40	5	難しさが増加した (5%の有意水準)

達成度を見ると、「その時間の課題を達成できなかった（達成度が 1)」と回答した生徒の割合は常に 10%未満と低く、最後まで授業が成立していたことがわかる。

- 授業が進むにつれて楽しさが増した

  楽しさを見ると、「楽しかった（楽しさが 3 か 4)」と回答した生徒の割合は、授業が進むにつれて 20%から 40%に増加した。これは、授業を前半と後半に分けて行った符合検定の結果からも有意な変化である。一方、「楽しくなかった（楽しさが 1)」と回答した生徒の割合は常に 10%未満で推移し、最後は 0%になった。

- 達成したときの喜びが大きかった

  難しさと楽しさを見ると、授業が進むにつれて難しいと感じる割合が増えているが、楽しさを感じる生徒の割合も増えている。最後の授業について達成度と楽しさの相関を求めたところ 0.55 であることから、課題の達成度が楽しさに結び付いていることがわかる。

- 難易度の高い時間があった

  難しさを見ると、タイマーを扱った 6〜8 時間目の難易度が高くなっている。6 時間目でタイマーの意味を学んだ後、7,8 時間目の応用に進むにつれて難しさを感じる生徒が増えていることから、タイマーを自分の作品に応用することが難しかったことがわかる。

5.2.2　2回の定期試験による評価

2回の定期試験を実施した。それぞれの試験において、生徒が理解している内容を計る項目を検討し、教員と相談しながら筆者が試験問題を作成した。

テスト 1（2学期末試験）

5時限までの内容を2学期の期末試験として出題し、評価を行った。出題のねらいと概念理解率を表5.3に、テスト問題を図5.4に示す。

表 5.3: 2学期末テストの問題と概念理解率

問題		概念理解率
1	逐次実行の理解	89.6%
2	プログラムの記述と繰り返しの概念	84.0%
3	構文エラーの発見	92.0%
4	プログラムの読解	91.2%
5	絶対座標の理解	73.6%
	平均	86.1%

採点の結果は、平均点が32.3点であり、満点（50点）が4人であった。概念理解率の平均が86.1%であることから、構文誤りまたは他の概念の誤りによる減点が約10点あったことがわかる。

生徒の理解度については、基本問題である問題1〜4の概念理解率が84%以上あることから、80%以上の生徒がプログラミングの基本概念を理解できていることが確認された。

以下に、各問題を考察する。

問1 逐次実行の理解

計算機の動作は、命令を順に実行していく逐次処理が基本である。この問題では、画面上のタートルオブジェクトにメッセージが順に送られるプログラムを示し、実行結果を図示させることで、生徒が逐次処理の概念を理解しているかどうかを調べた。

結果は、89.6%の生徒が「タートルオブジェクトに前進と回転の命令が交互に3回

第２学年　２学期　〔技術科　情報基礎「ドリトル」〕　2001.12.04

技術・家庭科　期末テスト問題

松阪市立鎌田中学校（筑波大学ドリトル開発チーム作成）

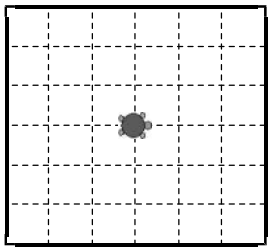
組
席

※解答は、すべて解答欄 ━━ にていねいな字で記入しなさい。氏名は、裏面に記入しなさい。

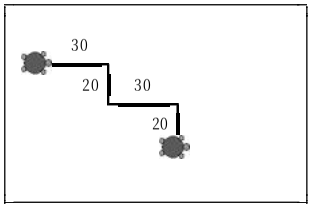
１．次のプログラムで、画面に描かれる線（かめたの線）を右の図に書きなさい。

```
かめた＝タートル！つくる。
かめた！１００ぽ　あるく　９０ど　みぎまわり。
かめた！１００ぽ　あるく　９０ど　ひだりまわり。
かめた！１００ぽ　あるく　９０ど　ひだりまわり。
```

※注意
右の回答欄の1目盛は100ぽぶんです　。



２．次のような「階段」を描きたいと思います。できるだけ長くないプログラムをかきなさい。
ただし、「１段の幅は３０ぽ、高さは２０ぽ」です。



プログラム

３．正方形を描こうと思って、次のプログラムを書きましたが動きません。何がまちがっているのかを説明しなさい。また、正しく直したプログラムを書きなさい。

```
かめた＝タートル！つくる。
「かめた！　みぎまわり　９０ど　１００ぽ　あるく」！４かい　くりかえす。
```

※まちがいの説明

※正しいプログラム

４．次の３つのプログラムのうち２つは、動かすと同じ形になります。どれとどれかを答えなさい。

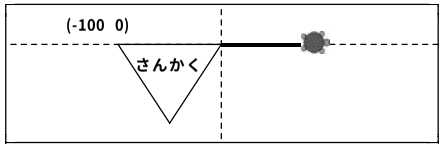
```
Ａ．かめた＝タートル！つくる　（青）ぬる。
　　さんかく＝「かめた！１００ぽ　あるく　１２０ど　みぎまわり」！３かい　くりかえす
　　　図形にする。

Ｂ．かめた＝タートル！つくる。
　　さんかく＝「かめた！１００ぽ　あるく　１２０ど　みぎまわり」！３かい　くりかえす
　　　図形にする。
　　さんかく！（青）ぬる。

Ｃ．かめた＝タートル！つくる。
　　さんかく＝「かめた！１００ぽ　あるく　１２０ど　みぎまわり」！３かい　くりかえす
　　　図形にする　（青）ぬる。
```

※解答　［　　］と［　　］は、同じ図形になります。

５．次の図のように三角形を移動させ、長さ１００の線を描くプログラムを作ろうと思いました。プログラムの空欄をうめて完成させなさい。



※解説
三角形は、画面の中心から、左へ100移動しています。
「かめた」は、画面の中心から、１００ぽ　動いています。
※ヒント
「移動する」命令の使い方
…（例）100 100 移動する

※プログラム

```
かめた＝タートル！つくる。
さんかく＝「かめた！１００ぽ　あるく　１２０ど　みぎまわり」！３かい　くりかえす
　　図形にする　（赤）ぬる。
```

６．「ドリトル」を使っての感想や要望を書きなさい。（質問もかいてもいいです）

図 5.4: ２学期末のテスト問題

ずつ送られる」という図を描くことができた。約 90%の生徒が逐次的な実行の流れを理解していることがわかる。

問 2 プログラム記述と繰り返し

プログラミング言語は決まった書き方の規則 (構文) を持ち、それに従ってプログラムを記述する必要がある。この問題ではタートルグラフィックスの実行結果を示し、その図形を描くためのプログラムを記述させることで、生徒がプログラムを記述できるかどうかを調べた。

また、設問の図形は階段の形をしており、いちどに 2 段を描くほかに、1 段ずつの描画を 2 回繰り返して描くことができる。問題では「できるだけ長くないプログラムをかきなさい。」という設問の形式をとることで、繰り返しを理解している生徒に繰り返しの存在を気付かせるようにした。

結果は、84.0%の生徒が階段状の図形を描くプログラムを記述した。また、半数の 42.0%の生徒が繰り返しを含むプログラムを記述した。約 85%の生徒が基本的なプログラムを記述することができ、そのうち半数の生徒が繰り返しを使いこなしていることがわかる。

問 3 構文エラーの発見

プログラムは、日常使用する自然言語と違い、使用する言語の構文に沿って厳密に記述する必要がある。構文に沿わないプログラムはエラーになり、正常な動作が行われない。この問題では、構文規則に適合しない記述を含むプログラムを示し、誤りの説明と修正後のプログラムを記述させることで、生徒がドリトルの構文を理解しているかどうかを調べた。誤った構文には、命令語と引数の順序が逆になった記述を使用した。

結果は、92.0%の生徒が “みぎまわり 90ど” が誤りであることを指摘し、正しく “90ど みぎまわり” と修正した[2]。90%以上の生徒が基本的な構文の規則を把握し、それに従うことが必要であることを理解していることがわかる。

[2]ドリトルの日本語命令は、“右回り” のような漢字表記を標準としている。今回の実験授業を行うにあたり、担当教員から「“塗る” のように自分が読めない難しい漢字が出現すると、生徒によっては学習意欲をなくしてしまう可能性がある」という指摘があり、“みぎまわり” のような仮名のみの命令表記を追加した。

問 4 プログラムの読解

ある動作を行うプログラムの記述は、一般に複数の書き方が存在する。この問題では、3つのプログラムを提示し、動作が同じ結果になる2つのプログラムを選択させることで、生徒がプログラムを正しく理解しているかを調べた。

選択肢とした3つのプログラム例は、メッセージ “ぬる” の使い方が異なっている。プログラム A では、タートルオブジェクトに図形用のメッセージ “ぬる” を送っているため、正しく動作しない。プログラム B とプログラム C は共にタートルグラフィックスの線分から生成した図形オブジェクトにメッセージを送る。プログラム B とプログラム C は、生成した図形オブジェクトを、いったん変数に入れてから使うか、変数に入れずにカスケード送信するかという使い方が異なるが、画面に描かれる図形は同一である。

結果は、92.2%の生徒が正解である B と C を選択した。90%以上の生徒がオブジェクトごとに解釈できるメッセージが異なることを理解していることがわかる。

問 5 絶対座標の理解

ドリトルのタートルグラフィックスでは向きと距離の指定によりオブジェクトを移動するが、そのほかにも絶対座標を指定した特定の座標への移動や、縦横の移動量を指定した相対的な移動が可能である。この問題では、図形とタートルオブジェクトを画面上に生成するプログラムと実行結果の画面を示し、最終的な位置に2つのオブジェクトを移動するプログラムを完成させることで、生徒がオブジェクトごとに適切な移動命令を使えるかどうかを調べた。

結果は、73.6%の生徒がタートルと図形を移動するプログラムを記述した。

この問題では、「図形とタートルの移動」という基本的な内容を扱っているにもかかわらず、他の問題と比較して正答率が低い結果となった。この問題は他の問題と比較して、次のような特徴を持っている。

- プログラムを途中まで示し、それを理解した上で続きを記述して完成させる。プログラムの読解と記述の両方の要素を含む問題である
- タートルだけではなく、生成した図形オブジェクトを移動する問題である (問題 1〜3 では図形を描くが図形オブジェクトは生成しない。問題 4 では生成し

た図形オブジェクトに色を塗るが移動は行わない)

- 図形の移動において、“さんかく！100 100 移動する。” (三角形を右に 100、上に 100 だけ移動する) のように、図形を数学的な座標で相対的に移動する操作を含む問題である
- 図中で三角形の頂点の位置を「(-100, 0)」で示すなど、数学的な座標の位置の概念を含む問題である

正答率が低くなった原因は、これらの特徴が影響していると考える。どの性質が強く影響しているかはこの結果だけでは明らかではないが、担当教員と検討した結果、絶対座標を使用していることの影響が大きいのではないかと考えている。他の問題ではタートルグラフィックスによる相対的な移動を扱っており、「100 歩進む」「90 度回転する」など座標の知識を必要としないのに対し、この問題では「(100, 100) の座標位置に移動する」など数学的な座標の知識を必要とするためである。今後、他の事例を含めて検討を進めたい。

テスト 2（3 学期末試験）

11 時限までの内容を 3 学期の期末テストとして出題し、評価を行った。出題のねらいと概念理解率を表 5.4 に、テスト問題を図 5.5 に示す。

表 5.4: 3 学期末テストの問題と概念理解率

問題	内容	概念理解率
1	オブジェクトの区別	90.2%
2	ボタンによる対話操作	92.7%
3	タイマーによる繰り返し	42.3%
4	仕様を満たすプログラム作成	74.8%
5	メソッドの定義	71.5%
	平均	74.3%

採点の結果は、平均点が 27.7 点であり、満点（50 点）が 2 人であった。概念理解率の平均が 74.3%であることから、構文誤りまたは他の概念の誤りによる減点が約 10 点あっ

第２学年　３学期　〔技術科　情報基礎「ドリトル」〕　　2002.2.27

技術・家庭科　学年末テスト問題

松阪市立鎌田中学校（筑波大学ドリトル開発チーム作成）

組
席

※解答は、すべて解答欄に、ていねいな字で記入しなさい。氏名は、裏面に記入しなさい。

問１　「タートル、図形、タイマー、ボタン」オブジェクトのうち、次の性質を満たすものを書きなさい。あてはまるオブジェクトを下のア～エが選び、記号で答えなさい。〔複数解答あり〕

ア．タートル（かめた）　イ．図形（さんかく）　ウ．タイマー　エ．ボタン　〔１０点〕

(1)押すことによって、指定した命令を実行することができる。

(2)「ひだりまわり」「みぎまわり」などで、回転することができる。

(3)間隔と時間を指定することにより、タートルや図形を、指定した間隔ごとに、指定した時間だけ動作させることができる。

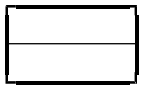
(4)「図形にする」で切り離すことによって、移動したり回転することができる。また、色を塗ることもできる。

(5)「あるく」で、前後に動かすことできる。

問２　ボタンを2つ作って、かめたを操縦する次のプログラムを作りました。

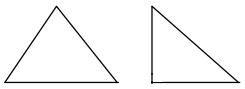
```
かめた＝タートル!つくる。
ボタン１＝ボタン!”あるく”つくる。
ボタン１！－２００　１００　位置　１５０　５０　大きさ。
ボタン１：動作＝「かめた！５０ぽ　あるく」。
ボタン２＝ボタン！”まわる”つくる。
ボタン２！０　１００　位置　１５０　５０　大きさ。
ボタン２：動作＝「かめた！９０ど　みぎまわり」。
```

(1)このプログラムでかめたを操縦して描くことができない図形はどれとどれでしょう。形があっていれば、大きさは自由です。〔２点〕

ア．　 

イ．　 

ウ．　 

エ．　 

(2)ボタン1かボタン2の動作のどちらかを手直しすれば(1)で回答したもの（描くことのできない図形）が描けるようになります。どちらのボタンをどう直せばよいでしょう。〔８点〕

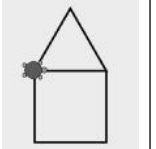
ボタン〔　　〕を次のように手直しすると描けるようになります。

問３　1秒ごとに赤い三角形が10ずつ右へ、青い三角形が10ずつ左へ動くプログラムを作ろうとしています。［　　］の中に命令を入れて完成させなさい。〔１０点〕

```
かめた ＝ タートル!つくる。
赤三角 ＝「かめた!100ぽ あるく 120ど みぎまわり」!3かい
　　くりかえす　図形にする　(赤)ぬる。
青三角 ＝「かめた!100ぽ あるく 120ど みぎまわり」!3かい
　　くりかえす　図形にする　(青)ぬる。
動き ＝ タイマー!つくる　1秒　間隔　10秒　時間。
動き!「　　」!実行。
```

問４　右図のような「家のかたち」をかめたに描かせようと思います。プログラムを作りなさい。やねの形は多少いびつでもかまいません。〔１０点〕

```
かめた ＝ タートル!つくる。
```



問５　ボタンを押すとかめたが動くプログラムを作りましたが、うまく動きません。次の(a)(b)の問いに答えなさい。〔１０点〕

```
かめた＝タートル！つくる。　　……１行目
ボタン１＝ボタン！”うごく”つくる。　　……２行目
ボタン１！－２００　１００　位置　１５０　５０　大きさ。　　……３行目
ボタン１：動作＝かめた！50ぽ あるく　９０ど　みぎまわり。　……４行目
```

(a) どう直せば動くようになるでしょうか。その行のプログラムを正しく書きなさい。

(b) なぜ(a)のように直すと動くのでしょうか。くわしく説明しなさい。
〔「○○がないから」「○○が間違っている」だけでは、説明不足です。どうして必要なのか、どうして間違いなのかまで、説明しなさい。〕

図 5.5: 3学期末のテスト問題

たことがわかる。

問題 1,2,4,5 の概念理解率から、タイマーを除くプログラム上の概念は 70%以上の生徒が理解していることが確認された。

以下に、各問題を考察する。

問 1 オブジェクトの区別

オブジェクトは、その種類ごとに、実行できるメソッドや画面上の表示、行える動作などの性質が異なる。この問題では、オブジェクトの性質を表す 5 種類の説明文を示し、当てはまるオブジェクトを 5 種類の選択肢から選択させることにより、生徒がオブジェクトごとの性質を理解しているかどうかを調べた。

結果は、90.2%の生徒が正しいオブジェクトを選択した。90%以上の生徒がオブジェクトごとの違いと基本的な性質を理解していることがわかる。

この問題では、複数個の選択を許したことから、厳密には性質を満さないオブジェクトを回答するケースがあった。たとえば（5）の設問では、“あるく” というメソッドを持つタートルオブジェクトが正解となる。しかし、図形オブジェクトも “移動する” といった類似のメソッドにより移動することが可能であることから、今回の評価では、図形オブジェクトを含めて回答した場合でも「基本概念を理解している」と判断した。

問 2 ボタンによる対話操作

プログラムは記述内容によって、行える動作の範囲が決まる。この問題では、「前進・回転という 2 つのボタンを操作して図形を描くプログラムを示し、描くことのできる図形を選択させる」問題により、生徒が与えられた制約下で可能な動作を判断できるかどうかを調べる。次に、「そのままでは描けない図形を描くためにプログラムを修正させる」問題により、生徒が与えられた制約をどう変更すれば可能にできるかの理解を調べる。提示したプログラムでは、回転する角度は 90 度になっている。そのため、90 度の倍数以外の角度を持つ図形は描くことができない。

結果は、92.7%の生徒が描ける図形として 90 度の倍数の角度を持つ頂点から構成される図形を選び、三角形などを描くために回転するボタンのメソッドを 15 度など汎用性の高い角度に修正することができた。90%以上の生徒が、ボタンの機能を修正することでプログラムの汎用性を増やせることを理解していることがわかる。

問 3 タイマーによる繰り返し

タイマーは、一定間隔で繰り返し実行を行うオブジェクトである。この問題では、画面に描かれた 2 つの図形をタイマーにより移動するプログラムの断片を示し、足りない部分を埋めてプログラムを完成させることにより、生徒がタイマーを理解しているかどうかを調査した。

結果は、42.3%の生徒がタイマーにより図形を動かす記述を行った。

この問題では、「図形のタイマーによる移動」という基本的な内容を扱っているにもかかわらず、他の問題と比較して正答率が低い結果となった。この問題は他の問題と比較して、次のような特徴を持っている。

- プログラムを途中まで示し、それを理解した上で続きを記述して完成させる。プログラムの読解と記述の両方の要素を含む問題である
- タートルだけではなく、生成した図形オブジェクトを移動する問題である
- 2 つの図形オブジェクトを同時に扱う問題である
- タイマーを扱う問題である。ただし、タイマーの構文は設問に次のように示されており、構文の暗記を前提としていない。

  動き！「 [　　] **」実行。**

- 図形の移動において、“赤三角！10 0 移動する。” (三角形を右に 10、上に 0 だけ移動する) のように、図形を数学的な座標で相対的に移動する操作を含む問題である

正答率が低くなった原因は、これらの特徴が影響していると考える。どの性質が強く影響しているかはこの結果だけでは明らかではないが、他の問題と比較して正答率が悪かった理由を検討したところ、担当教員からは「12 月にタイマーを扱う授業を行った後、1,2 月に別の機能 (ボタンを扱う授業) を行っていたため、試験前の 2ヵ月間に授業の中でタイマーを使う機会がなかったせいではないか」という回答があった。今後、他の事例を含めて検討を進めたい。

問 4 仕様を満たすプログラム作成

ある動作を行うプログラムの記述は、一般に複数の書き方が存在する。この問題で

は、三角形と四角形を組み合わせた家の形を描くプログラムを課題として提示し、その図形を描くプログラムを作成させることにより、生徒が与えられた目標を満たすプログラムを計画を立てて作成できるかどうかを調査した。

この課題を満たすプログラムは、描く順番 (四角形を描いてから三角形を描く、三角形を描いてから四角形を描く)、描く向き (右向きに描いていく、左向きに描いていく)、線分か図形か (一筆書きで図形を作らずに描く、三角形と四角形の図形オブジェクトを作り組み合わせる) など、数多くの作成方法が考えられる。

結果は、74.8%の生徒が課題の図形を描くプログラムを作成することができた。作りたいプログラムを思い描いたときに、70%以上の生徒が具体的なプログラムの形で記述できることがわかる。

問 5 メソッドの定義

ドリトルのメソッドは、オブジェクトのプロパティにブロックを代入する形で定義する。ブロックには後から実行したいプログラムコードが記述されており、それをプロパティに入れて保存することにより、後から呼び出して使用することができる。この問題では、メソッド定義の誤った構文を示し、修正後のプログラムと誤りの理由を記述させることにより、生徒がメソッド定義の構文を理解しているかどうかを調査した。

ドリトルのメソッドはブロックをプロパティに代入する形で定義するが、この問題ではブロックを示す記号 (「…」) を記述していないため、メソッドが定義されない。

ボタン1:動作=かめた!50ぽ あるく 90ど みぎまわり。

結果は、71.5%の生徒が次のように正しくプログラムを修正した。

ボタン1:動作=「かめた!50ぽ あるく 90ど みぎまわり」。

なぜ括弧 (「…」) を付けると動作するのかという説明は、授業の中でブロックについて説明しなかったこともあり、ほとんどの生徒は「括弧を付けないとボタンの動作を定義できないから」という構文上の規則として回答した。2 名だけ「括弧を付けないとその場で実行されてしまう」のように、「プログラムを内包するオブジェクト」であるブロックの存在に気付いていると思える回答を行った。

5.2.3　授業実施後のアンケートによる評価

ドリトルを学習した本人が「理解できている」と感じているかどうかを調べるために、今回の授業で扱ったプログラミングやオブジェクト指向の内容について、生徒に「自分が理解していると思うか」という主観的なアンケートを実施した。

結果を表 5.5 に示す。表中で「不明」は「設問の意味がわからなかった」という選択肢である。

以下に、アンケートから読み取れる結果を示す。

- 授業の中で説明を行った設問 (1)〜(4) は、肯定的な回答（1 または 2）が 60%以上あり、プログラミングの基礎概念を理解できたと感じている生徒が多い。
- 授業で説明していない設問（×）のうち、(5), (6) については約 60%の生徒が「プログラムを正しく書けば自分の考えた通りに動く。エラーがあれば動かない」と経験的に認識している。
- （7）については、授業で説明していないにも関わらず、「タイマーが非同期に実行される」ことに気付いている生徒が約 50%いる。
- (8) については授業で説明していないこともあり、設問を理解できない生徒が 33%存在した。ブロックの存在は授業の中で説明しない限り理解することは難しいことがわかる。

5.2.4　作成したプログラムの分析

9〜11 時間目の授業では、学習内容をまとめる課題として、ボタン操作で線や図形を描くペイントソフトを作成した。

図 5.6 に、生徒作品の例を示す。このプログラムでは、画面左にボタンが配置されており、それぞれのボタンに次の種類のメソッドが定義されている。

- タートル操作ボタン。タートルを操作する（移動、回転、ペンの上げ下げ）
- 図形描画ボタン。繰り返し命令でタートルを移動し、図形オブジェクトを生成する
- 色指定ボタン。ペンの色を変更する

表 5.5: 理解度アンケートの結果

設問		授業	肯定 ←				→ 否定	不明
		説明	1	2	3	4	5	X
(1)	プログラムの書き方はきっちり決まった規則があって、その規則を少しでもはずれていると動作しない。	○	60.0%	21.7%	9.6%	5.2%	1.7%	1.7%
(2)	「あるく」「移動する」「塗る」などの命令の前には、「どれくらい動くか」「どこに移動するか」「何色に塗るか」などの指定をつけることができる。	○	62.6%	14.8%	10.4%	2.6%	4.3%	5.2%
(3)	「タートル」「図形」「タイマー」「ボタン」など、ものの種類ごとに見え方やできることが違っている。	○	47.0%	23.5%	12.2%	3.5%	2.6%	11.3%
(4)	さまざまなものに命令するときは、そのものを指定した後に「!」をつけて、それから命令を書く。	○	38.9%	25.7%	18.6%	4.4%	7.1%	5.3%
(5)	プログラムは原則として書かれた順番どおり上から下、左から右の順番にきっちり実行されていく。	×	33.9%	26.1%	17.4%	6.1%	1.7%	14.8%
(6)	コンピュータは自分が書いたプログラムの通りに厳密に動作するようにできている。	×	27.0%	30.4%	26.1%	7.8%	3.5%	5.2%
(7)	タイマーによる動作だけは(5)と異なり、タイマーに指示したタイミングで実行される。	×	27.2%	21.1%	21.1%	7.0%	0.9%	22.8%
(8)	繰り返し実行させる動作、タイマーに実行させる動作、ボタンが押されたときの動作などは「…」という形で囲んだもの(ブロック)で指定する。	×	18.3%	17.4%	20.9%	4.3%	6.1%	33.0%

- 姿の指定。タートルの画像を変更する

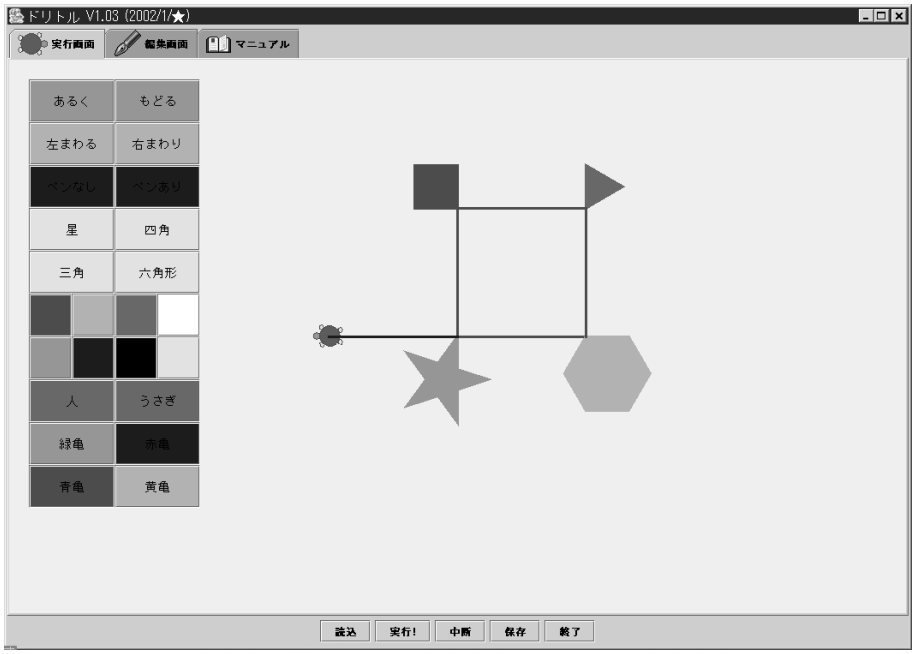


図 5.6: 生徒作品（ペイントソフト）

学んだ知識を活用することで、ほぼ全員が豊富な表現力を持つペイントソフトを作成することができた。

表 5.6 に、「ペイント」を含む各種の生徒作品の中で使われたプログラミングの概念を示す。ボタンやタイマーを始めとする高度な概念を、85%以上の生徒が活用していることを確認できた。

表 5.6: 生徒作品の分析

機能	使用率
タートル	100%
図形オブジェクト	97%
繰り返し	96%
ボタン	93%
タイマー	85%

5.3 考察

実験授業で実施した「授業ごとのアンケート」「2 回の定期試験」「理解度アンケート」「作成されたプログラム」の結果を分析して得られた知見は次のものがある。

- 中学校において、オブジェクト指向言語を用いた情報教育が可能であることを確認した。定期試験の分析と生徒作品の分析から、11 時間という短い時間の中で、繰り返しやメソッド呼び出しなどのプログラミングの基本概念のほか、ボタンへの動作メソッドの定義やタイマーを用いたスレッドなどを扱えることを確認した。

- 学年全体（4 クラス 132 名）を対象とした集合教育の中で、プログラミングの授業が成立した。定期試験の分析から、約 9 割の生徒がプログラミングの基本概念を理解していることを確認した。毎時間のアンケートの分析から、授業が進むにつれて楽しさが増し、達成感につながっていることを確認した。

- プログラミングを学ぶ上で、オブジェクト指向の概念が特に問題になることはなかった。概念理解度のアンケート分析では、オブジェクトの存在（設問 3）とメッセージ送信（設問 4）について、65%以上の生徒が肯定的な回答（「1」または「2」）をしていることから、図形をオブジェクトとして位置付け、色を塗ったり位置を移動したりできるという形でのオブジェクト指向の利用は、生徒にとって抵抗なく受け入れられたと言える。また、毎時間のアンケートの分析では、オブジェクトの生成を扱った授業（たとえば複製によりタートルオブジェクトを生成した 1 時間目、描いた軌跡から図形オブジェクトを生成した 3 時間目など）に難易度が増す傾向は見られないことから、クラス定義を用いずにオブジェクトを複製によって生成するプロトタイプ方式の採用は、オブジェクト指向言語に対する初期の敷居を下げることができたと考える。

授業実践を通して得られた知見は次のものがある。

- GUI 部品（特にボタン）を学び、「ボタンを押すと何らかの動作が起きる」形のプログラムを作成することに多くの生徒が強い興味を示した。授業を担当した井戸坂教諭は、「GUI 部品を利用したペイントソフト等の作品により、生徒たちは実社会で利用されているソフトウェアと、自分たちが学んだプログラミング学習を結び付けることができた」[47] と観察している。一般の言語において、GUI 部品を利用した

プログラムの作成は容易でないことが多いが、今回の実験授業では、プログラミングを学び始めて数時間の生徒がボタンを使ったプログラムを作成できた。普段使っている「マウスでクリックしながらアプリケーションを対話的に操作する」形のプログラムを作ることで、生徒は自分たちが価値のあることを学んでいることを実感し、実用的なプログラムを作れるという自信につながったと考える。

- タイマーを用いたアニメーションを、授業の中で学習できた。ただし、毎時間のアンケートの分析から、タイマーは 6〜8 時間目に扱う内容としては難易度が高いことがわかった。9 時間目以降に実施したボタンオブジェクトをタイマーより易しいと感じる生徒が多いことから、今後はボタンオブジェクトで対話的な操作を学んだ後でタイマーによる自動的な繰り返しを学んだり、タイマーの学習に費やす時間を増やすなど、カリキュラムの改善を行うことが有効と考えている。

5.4 情報教育とプログラミングに関するまとめ

プログラミング経験のない生徒を対象に、中学校でドリトルを用いた実験授業を行った。授業は正規の教科 (技術科) の中で行われ、2 学期間に渡り、学年全体 (4 クラス 132 名) の生徒が受講した。評価のために、2 回の定期試験、毎時間のアンケート、理解度のアンケート、作品プログラムの分析により、評価を行った。

その結果、中学校においてオブジェクト指向言語を用いた授業が可能であることを確認した。11 時間という短い時間の中で、生徒はボタンやタイマーによるスレッドを活用したプログラム作品を作ることができた。授業が進むにつれて扱う題材の難易度は高くなったが、生徒の興味や楽しさが失われることはなく、逆に楽しさや達成感が増した。

一般に、難易度が易しいと授業が成り立つが、難しくなると付いて行けない生徒の割合が増加する。しかし、今回の実験授業では、授業が進むにつれて難易度が上がっても、付いてくる生徒が多く、楽しさの失われない授業となった。これはプログラミングという題材の力であると同時に、ドリトルがプログラミングの魅力を伝えることのできる言語であったためと考える。

第6章　分散プログラミングと情報教育

近年ではインターネット環境の普及が急速に進んでいる。平成15年に公開された総務省の統計 [60] によると、平成9年末にわずか6.4%だったインターネットの世帯普及率[1]は、平成14年末現在で80%を超え、特に13〜29歳の若年層のインターネット利用率[2] は90%近くに達している。

このような背景から、今後の情報教育においては、スタンドアロンの計算機環境だけでなく、日常接する情報通信の中で計算機の果たす役割を理解することが重要である。

そこで、本章ではドリトルを拡張した「分散共有ドリトル」を実現することにより、ネットワーク上の分散環境でオブジェクトを活用する学習環境を提案する。

6.1　情報教育と分散プログラミング

6.1.1　情報教育における情報通信

情報教育の中で分散プログラミングを利用するためには、使用するプログラミング言語にどのような性質が求められるだろうか。本節では、情報通信の仕組みを扱う教科として中学校の「技術・家庭科」と高等学校の「情報B」を取り上げ、学習指導要領 [75] [77] の検討を通して、教育用のプログラミング言語に求められる性質を考察する。

中学校「技術・家庭科」[77] の技術分野である「情報とコンピュータ」では、「情報通信ネットワークについて、情報の伝達方法の特徴と利用方法を知ること」など、情報通信の仕組みを含む内容が扱われている。情報通信における情報の伝達方法の特徴を知るためには、情報が計算機のデータで表現され、それが計算機同士の通信により伝達されること

[1]パーソナルコンピュータや携帯電話等の電子機器により、WWWの閲覧や電子メールを使用している個人のいる世帯の比率を示している。

[2]パーソナルコンピュータのほか、インターネット対応型の携帯電話・PHSの利用を含む。また、自宅での利用のほか、学校・職場での利用も含まれている。

を理解できる環境が必要である。

高等学校「情報 B」[75] では、「情報通信と計測・制御の仕組み及び社会におけるそれらの技術の活用について理解させる」とし、「動作を確認できるような学習を取り入れるようにする」など、情報通信の仕組みを含む内容が扱われている。情報通信の仕組みを動作を確認しながら学ぶためには、計算機同士が一定の手順に従って情報を伝達していることを理解できる環境が必要である。

6.1.2 分散共有ドリトルに求められる特徴

中学校と高等学校における以上の学習指導要領の内容を考慮し、情報通信の学習に利用できる分散共有ドリトルの性質を検討する。

分散共有ドリトルでは、新たにネットワーク上でオブジェクトを扱う機能を提供することで、個人ごとのプログラミングだけでなく、他の生徒とコラボレーション作品を作成したり、ネットワークでの通信をプログラムを通して体験することを可能にする。

分散共有ドリトルの設計にあたっては、ドリトルの入門用言語としての利点を維持する形で拡張を行った。設計にあたり留意した点を示す。

- ネットワークプログラミングの未経験者が扱えること
  中学校での利用を想定すると、分散共有ドリトルを使う生徒と教員に、ネットワークの専門知識やネットワークプログラミングの経験を期待することはできない。よって、ドリトルを使った経験のある生徒と教員であれば、マニュアルを読みながら使える程度の難易度とする。

- プログラミングを通して、日常接する情報通信の原理を類推できること
  電子メールや WWW など、生徒が日常接する情報通信では、文字や画像などのデータが相互に転送されることで実現されている。分散共有ドリトルにおいては、文字列やテキストボックスにより文字を転送したり、タートルや図形オブジェクトにより画像を転送することで、計算機同士がデータを交換するモデルを体験的に学ぶことを可能にする。

- ネットワークプログラミングや分散プログラミングを学ぶことは目的としない
  ドリトルがプログラミングの学習を目的とせず、プログラミングを通して計算機 (特

にソフトウェア) の動作原理を体験的に学ぶことを目的とするように、分散共有ドリトルもネットワークプログラミングの学習を目的とせず、プログラミングを通してネットワークにおける情報通信の原理を体験的に学ぶことを目的とする。

- 情報通信のすべての原理を学ぶことは目的としない
  情報通信の原理は多岐にわたっており、ひとつの学習環境ですべてを網羅することはできない。分散共有ドリトルでは、ネットワーク構築の代表的な目的 [57] のうち、「計算機間の通信」、「分散処理」を扱う。「計算機資源の共有」、「人間の通信の媒体」については、計算機のファイル共有機能、電子メールの体験など、他のツールにより体験することが可能である。「計算機間の通信」については、計算機が扱うデータを転送すること、計算機同士があらかじめ決めておいた手順 (プロトコル) で通信を行うことなどを体験できるようにする。ただし、実際の情報通信で使われている複数の階層で構成されたプロトコルスタックなどについては、適宜講義による補足説明を行うことが望ましい。「分散処理」については、複数の計算機が協調して、計算などひとまとまりの動作を行うモデルを体験できるようにする。ただし、実際の分散処理で必要になる同期処理 [45] などについては、適宜講義による補足説明を行うことが望ましい。

- ドリトルのオブジェクトモデルを維持する
  ドリトルでは、画面上のオブジェクトにメッセージを送り、結果を視覚的に確認する形でプログラミングを行う。分散共有ドリトルでも、画面に表示されたネットワーク上のオブジェクトに対して操作を行うことで、結果を視覚的に確認しながらプログラミングを行うモデルとする。

6.1.3 既存言語における分散共有機能の検討

分散共有ドリトルの設計にあたり、分散機能を持つ既存の言語について検討した。

Java[13] はネットワークでの利用を考慮して設計されたオブジェクト指向言語である。Java には RMI(Remote Method Invocation) をはじめ、Jini[36]、CORBA(Common Object Request Broker)[25]、HORB(Hirano Object Request Broker)[14] など各種の分散オブジェクト指向環境が提供されている。しかし、教育を想定した言語ではないため、初心者が利用するには敷居が高いという問題がある。

スクリプト言語である Ruby[72] を分散環境に拡張したプログラミング環境として、dRuby[59] がある。dRuby は Ruby で記述されており、リモートに存在する Ruby オブジェクトのメソッド呼び出しを可能にする。図 6.1 に dRuby のサンプルを示す。この例では、サーバーに "druby://myhost:8470" で接続し、サーバー上の counter オブジェクト (数の数え上げを行うオブジェクト) にいくつかのメッセージを送信している。サーバー上のオブジェクトにメッセージを送り実行する dRuby のモデルは分散共有ドリトルと共通する部分が多いが、汎用言語である Ruby が画面に表示されないオブジェクトを基本とするのに対し、ドリトルでは画面上に表示されるオブジェクトを中心に操作する点が異なっている。

```
require 'drb/drb'

DRb.start_service

uri = "druby://myhost:8470"
counter = DRbObject.new(nil, uri)
p counter.up
p counter.up
counter.reset
p counter.up
```

図 6.1: dRuby のプログラム例 (文献 [59] より引用して改変)

6.2 分散共有機能

6.2.1 基本的な考え方

ドリトルでは、画面上のオブジェクトにメッセージを送り、操作の結果を視覚的に確認するモデルにより、初心者がオブジェクトを単位としたプログラミングを行える。また、グラフィックスや GUI 部品を使用したプログラミングを体験することを通して、日常接するさまざまな計算機の中でプログラムが動作していることを学ぶことができる。

ドリトルのオブジェクトは、変数 (プロパティ) や配列に格納し、取り出して使うことができる。分散共有ドリトルでは、オブジェクトの格納をネットワーク上に拡張する形で、初心者がオブジェクトの転送を容易に扱えるようにした。分散共有ドリトルのプログラミングを体験することを通して、日常接するさまざまなデータ通信の中で、プログラムが動作していることを学ぶことができる[3]。

分散共有ドリトルの実行時には、ネットワーク上にオブジェクトサーバーのプロセスを起動しておく。オブジェクトサーバーには、名前を付けてドリトルのオブジェクトを登録することができる。登録したオブジェクトは、複製と共有という 2 通りの活用を行えるようにした。

オブジェクトの複製 (図 6.2) は、サーバーからオブジェクトの複製を取り出す。取り出されたオブジェクトは通常のローカルなオブジェクトとなる。この機能はサーバーを介して複数のドリトル環境 (クライアント) 間でオブジェクトや値をやりとりするために用いる。

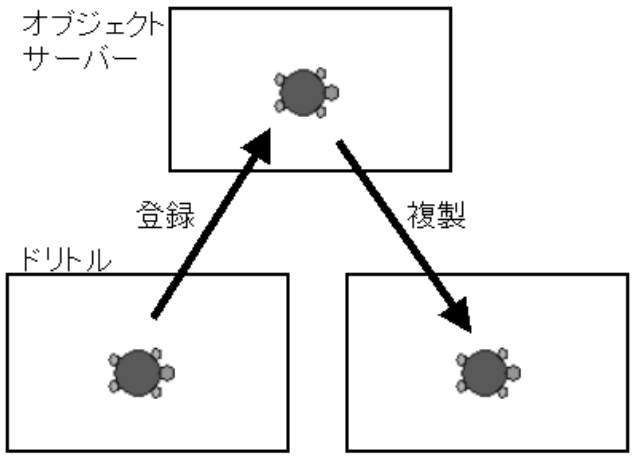


図 6.2: オブジェクトの登録と複製

オブジェクトの共有 (図 6.3) は、サーバー上のオブジェクトを共有する形で取り出す。この場合、取り出されるのはサーバー上のオブジェクトを参照する「共有オブジェクト」となり、共有オブジェクトに対する操作は、そのサーバー上のオブジェクトに対応するすべての共有オブジェクトに影響を及ぼす。

[3]分散共有ドリトルでは、「ネットワークプログラミング技術の習得」を主要な目的とはしていない。そこで、ソケット通信などの習得が難しいと思われるモデルは採用しなかった。

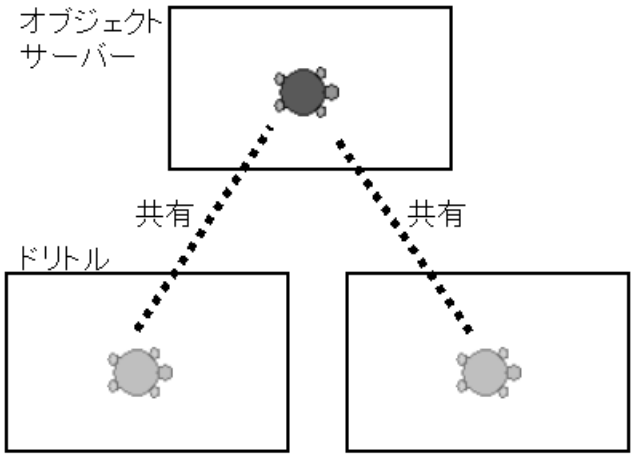


図 6.3: オブジェクトの共有

ドリトルのオブジェクトは画面上に見えるオブジェクトと見えないオブジェクトに大別される。見えないオブジェクトは内部的に値やメソッドを保持して動作するだけなので、見えないオブジェクトを共有する場合はサーバー上のオブジェクトが唯一の実体となり、取り出された共有オブジェクトはプロキシ[4]となってすべての操作をサーバーに中継する。

一方、見えるオブジェクトの場合は、オブジェクトの変化が各クライアントでの画面の変化として現れ、また画面上のオブジェクトへの操作がサーバー上のオブジェクトに伝えられる必要がある。このため、サーバー上のオブジェクトが MVC モデル[5] [10] における Model に、各共有オブジェクトが View+Controller になるように切り分けを行った。Model に対する変更はデザインパターン [9] のひとつとして知られている Observer パターンと同様の仕組みで View+Controller に伝達されるが、これをプログラミング初心者が理解し制御するのは難しいと考えたので、サーバーから共有オブジェクトを取り出すだけで View+Controller の生成と Observer の設定が自動的に行われるようにしている。

以下では、登録、複製、共有の機能を順に説明する。

[4]他のオブジェクトの代理となってメッセージを受け付けるオブジェクト。

[5]Model-View-Controller。ウィンドウシステムのような画面に表示されるオブジェクトを表現するために Smalltalk[12] で使われた。扱いたいオブジェクトを、実体 (Model)、Model を表示するためのオブジェクト (View)、ユーザからの入力を受け付けるオブジェクト (Controller) に分割して扱う。

6.2.2 オブジェクトの登録と複製

図 6.4 に、オブジェクトサーバーにオブジェクトを登録・複製するプログラム例を示す。

```
// ローカルオブジェクトの生成 (1)
カメ太=タートル!作る。

// サーバーへの接続 (2)
サーバー!"sv1" 接続。

// オブジェクトの登録 (3)
サーバー!"kame1"（カメ太）登録。

// オブジェクトの複製 (4)
カメ吉=サーバー!"kame1" 複製。

// オブジェクトの操作 (5)
時計=タイマー!作る
時計!「カメ太!10 歩く。カメ吉!15 歩く」実行。
```

図 6.4: オブジェクトの登録と複製

あるクライアント A で実行することを考える。

(1) で、タートルオブジェクトを生成する。

(2) で、サーバー “sv1” に接続する。引数には IP アドレスまたはホスト名を指定できる。

(3) で、サーバーにオブジェクト “カメ太” を “kame1” という名前で登録する。

(4) で、サーバーから “kame1” を複製し、カメ吉という名前にする。

(5) で、画面上の 2 つのオブジェクト (カメ太とカメ吉) を同時に動かす。サーバーから複製したオブジェクトは、ローカルのオブジェクトと区別することなくメッセージを送り操作することが可能である。

6.2.3 オブジェクトの共有

図 6.5, 図 6.6 にプログラム例を示す。

```
// サーバーへの接続 (1)
サーバー!"sv1" 接続。

// オブジェクトの共有 (2)
カメ助=サーバー!"kame1" 共有。
```

図 6.5: オブジェクトの共有 (1)

```
// サーバーへの接続 (1)
サーバー!"sv1" 接続。

// オブジェクトの共有 (2)
カメ助=サーバー!"kame1" 共有。

// オブジェクトの操作 (3)
時計=タイマー!作る
時計!「カメ助!10 歩く」実行。
```

図 6.6: オブジェクトの共有 (2)

クライアント B で図 6.5 を、クライアント C で図 6.6 を実行することを考える。

(1) で、サーバー “sv1” に接続する。

(2) で、サーバー上のオブジェクト “kame1” を共有し、“カメ助” というローカルの名前にする。

(3) で、クライアント C から共有オブジェクトを操作する。クライアント C の共有オブジェクトに送られたメッセージはサーバー上のオブジェクトに転送され、実行が行われる。サーバー上のオブジェクトの状態が変化すると、クライアント B, クライアント C の共有オブジェクトも画面上で位置の変化などを反映する。

6.3 実装

6.3.1 分散共有ドリトルの構成と動作環境

図 6.7 に分散共有ドリトルの構成を示す。ネットワーク上にオブジェクトサーバーを置き、ドリトルのオブジェクトを格納する。オブジェクトサーバーは画面を持ったプロセスである。ドリトルと同一のホスト上で動作してもよく、また、ネットワーク中に複数存在してもよい[6]。

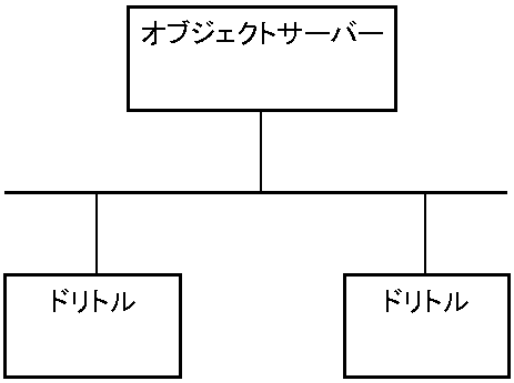


図 6.7: 分散共有ドリトルのシステム構成

図 6.8 にオブジェクトサーバーの実行画面を示す。モニター画面には、サーバーに登録されたオブジェクトが表示される。この例では、3 種類のオブジェクト (図形、タートル、ボタン) が表示されている。

オブジェクトサーバーはドリトルと似た画面インターフェースを使用しているが、プログラムを編集・実行するインターフェースは用意していない。また、画面上のオブジェクトは表示されるだけであり、画面上のボタンを押してもオブジェクトにメッセージは送られない。

サーバーに登録されたオブジェクトの一覧は、クライアント側のドリトルから確認できる。図 6.9 に、サーバー上のオブジェクトを表示するダイアログを示す。ダイアログ内には、「名前 (種類):補足情報」が表示される。名前はサーバーに登録されたオブジェクト名であり、種類はオブジェクトの種類を示す。補足情報には、可能であれば名前以外のオブ

[6]授業の中で、グループごとにサーバーを立てることができる。ただし、現在のバージョンでは、ひとつのプログラムから複数のサーバーに接続する使い方は想定していない。

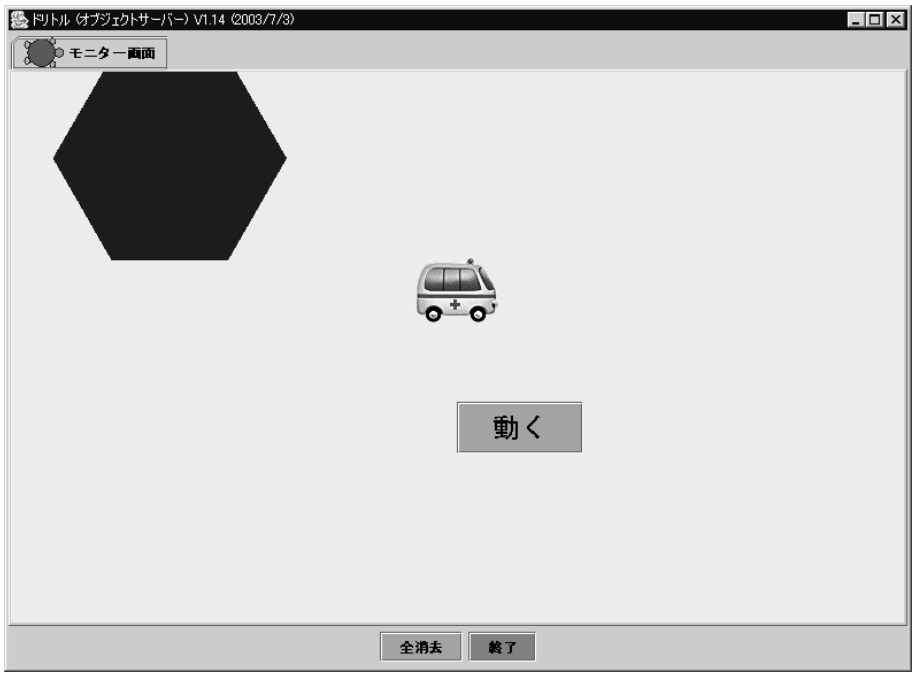


図 6.8: オブジェクトサーバーの実行画面

ジェクトを示す性質を表示する。図の 2 行目では、補足情報としてボタンのラベル名が表示されている。

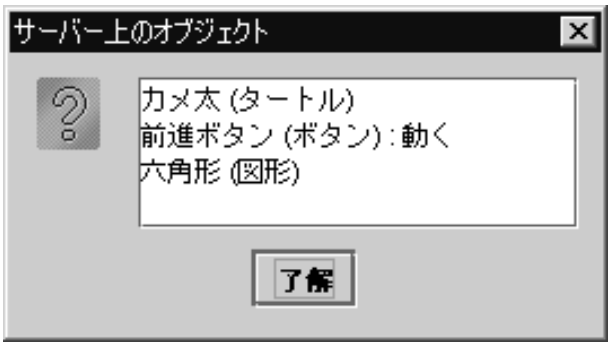


図 6.9: オブジェクトリストの表示

6.3.2 ネットワークへの拡張

図 6.10 に、ネットワーク経由でのメッセージ実行を示す。通常の (ローカルな) オブジェクトに対するメッセージ送信は直接ローカルな環境内で実行される。これに対し、共有オブジェクトに対するメッセージ送信はすべてサーバー上の対応するオブジェクトに転送され、そこで実行される。すなわち、共有オブジェクトはプロキシとして動作する。

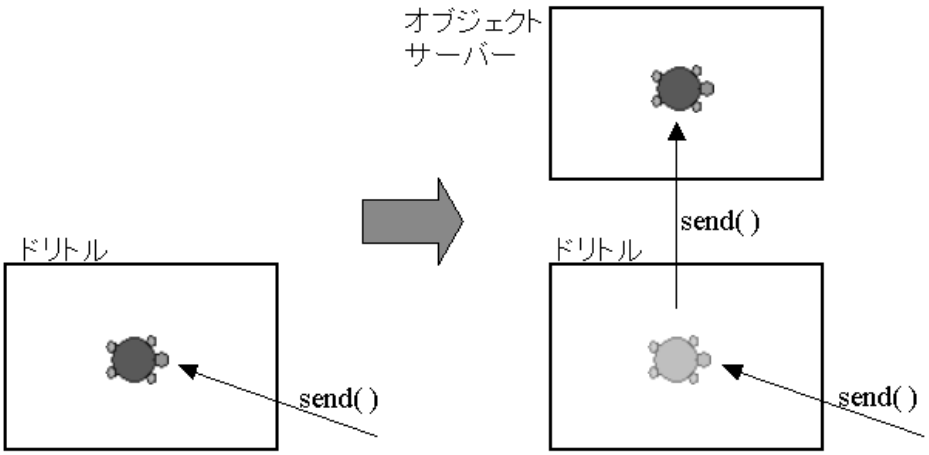


図 6.10: ローカル実行とリモート実行

サーバー上のオブジェクトは複数のクライアントによって共有可能である (図 6.11)。共有オブジェクトの 1 つにメッセージが送られると、実際の動作はサーバー上のオブジェクトが行い、サーバー上のオブジェクトの状態が変化する。この変化は他のクライアントから (その共有オブジェクトを通じて) 観測可能であり、これにより情報の交換が行える。

図形など画面に見えるオブジェクトの場合、共有オブジェクトは View+Controller となり、画面表示、クリックや衝突のイベントを処理する。オブジェクトの位置や色は Model であるサーバー側のオブジェクトが保持する。これらを扱うメソッドについては共有オブジェクトはプロキシとして動作する。Model の状態が変化すると Observer パターンによるコールバックが実行され、共有オブジェクトはそれに応じて画面表示などを更新する。

表 6.1 に、複製・共有可能なオブジェクトを示す。ドリトルのオブジェクトは原則として登録・複製可能だが、機器の入出力や通信を担うオブジェクト (サーバー、シリアルポート) は除外している。また、共有は状態を持たない (immutable な、不変な) オブジェクトに対しては複製と区別できないため、そのようなオブジェクトに対して共有を指示した場合は単に複製を返すようになっている (表 6.1 で左の欄が○、右の欄が×のもの)。

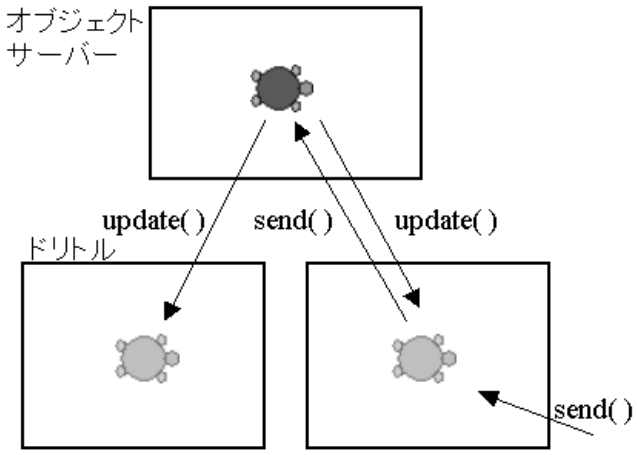


図 6.11: オブジェクトの共有

表 6.1: 複製・共有可能なオブジェクト

オブジェクト	登録・複製	共有	オブジェクトの性質
タートル	○	○	
図形	○	○	
GUI 部品	○	○	
配列	○	○	
タイマー	○	○	
数値	○	×	immutable(不変)
文字列	○	×	immutable(不変)
論理値	○	×	immutable(不変)
色	○	×	immutable(不変)
ブロック	○	×	immutable(不変)
サーバー	×	×	I/O(入出力)
シリアルポート	×	×	I/O(入出力)

以下では、図 6.4〜図 6.6 のサンプルを使い、接続、登録、複製、共有、共有実行の内部動作を説明する。

6.3.3 接続

サーバー！”sv1” 接続。

ローカルの “サーバー” オブジェクト (OxStore) は、“接続” メッセージを受け取ると、次の手順でサーバー側のオブジェクト (RemoteStoreServer) と接続する。

- 次の形式でサーバーを lookup する。
  `rmi://`ホスト`:`ポート`/bind`名
- 得られたサーバーオブジェクト (RemoteStoreServer) の参照をローカル変数 store に保持する

ここで、ホストは “接続” メッセージに引数として指定された値である。引数が省略された場合は “localhost” が使われる。ポートは、ドリトルの初期化ファイルに定義された値を使う。デフォルトは “2020” である。bind名は、ドリトルの初期化ファイルに定義された値を使う。デフォルトは “d0” である。

サーバー側では、接続を受け付けるオブジェクト (RemoteStoreServer) を bind して待機する。

6.3.4 登録

サーバー！”t1” (カメ太) 登録。

クライアントの “サーバー” オブジェクトは、“登録” メッセージを受け取ると、次の手順でサーバーにオブジェクトを登録する。

- オブジェクトを前処理する
- オブジェクトのプロパティを平滑化する
- サーバーに登録する

- オブジェクトを後処理する

前処理では、Javaで直列化 (serialize) できない要素を直列化可能なオブジェクトに変換する処理を行う。この処理の例としては、タートルオブジェクトの画像情報である Java の Image クラスのオブジェクトを ImageIcon クラスのオブジェクトに変換する処理と、図形オブジェクトの幾何学情報である GeneralPath クラスのオブジェクトを転送用の配列データに変換する処理がある。直列化できないデータは transient として宣言されており、サーバーには転送されない[7]。

プロパティの平滑化では、転送するオブジェクトに親オブジェクトのプロパティを格納する処理を行う。平滑化の詳細は後述する。

登録処理では、サーバーオブジェクトに対して put() メソッドを実行することにより、サーバーに名前とオブジェクトを転送し、オブジェクトを登録する。

オブジェクトの後処理では、前処理で生成した転送用のデータをクリアする。

サーバー側では、次の手順で登録を行う。

- 登録名とオブジェクトを名前テーブル (objmap) に登録する
- オブジェクトに後処理をする
- オブジェクトに親オブジェクトを登録する
- ドリトルの広域変数として、登録名でオブジェクトを登録する

名前テーブルは、サーバーに登録されたオブジェクトを管理するテーブルである。名前からオブジェクトを参照する辞書を提供する。名前テーブルへの登録では、登録名でオブジェクトを登録する。すでに同名のオブジェクトが存在した場合には上書きされる。

オブジェクトの後処理では、転送用に加工されたデータを戻す処理を行う。具体的には、転送用のデータからタートルオブジェクトの画像と図形オブジェクトの幾何学データを復元する。この処理により、登録したオブジェクトがサーバー画面に表示される。

登録したオブジェクトの親オブジェクトを設定する。親オブジェクトには、存在すればプロトタイプオブジェクトを設定し、そうでない場合はルートオブジェクトを設定する。この処理により、共有時にサーバー上でオブジェクトを実行できる。

[7] Java で transient 修飾子のついたインスタンス変数は直列化されず、復元されたオブジェクトにおいてそのようは変数の値は null になる。

広域変数への登録では、サーバー上のドリトルの名前空間にオブジェクトを登録する。この処理により、共有時にサーバー上でオブジェクトを実行できる。

ドリトルのオブジェクトは、親リンクとプロパティにより、他のオブジェクトとのリンク関係を持つ。あるオブジェクトをサーバーに登録するとき、転送される範囲は次のように決定される。

- プロパティは転送する。
- 親オブジェクトは転送せず、代わりに平滑化を行う。

  平滑化とは図 6.12 に示すように、親オブジェクトのプロパティを集めて来て子オブジェクトに格納する操作である。平滑化を行うのは、ドリトルのようなプロトタイプ方式のオブジェクト言語では、オブジェクトのプロパティは概念的には親オブジェクトのプロパティまでを含めた全体であるため、これらを合わせて転送しなければ転送先でオブジェクトが正しく動作させられないためである。親が多段になっている場合にも平滑化は再帰的に行うが、システム標準のオブジェクト (ルートオブジェクトを含む) や共有オブジェクトが現れた場合はそこまでで止める。

6.3.5 複製

カメ吉＝サーバー！”t1” 複製。

クライアントの “サーバー” オブジェクトは、“複製” メッセージを受け取ると、次の手順でサーバーからオブジェクトを複製する。

- サーバーからオブジェクトを複製する
- オブジェクトを後処理する
- オブジェクトに親オブジェクトを登録する

複製処理では、サーバーオブジェクトに対して getCopy() メソッドを実行することにより、サーバーからオブジェクトを転送する。

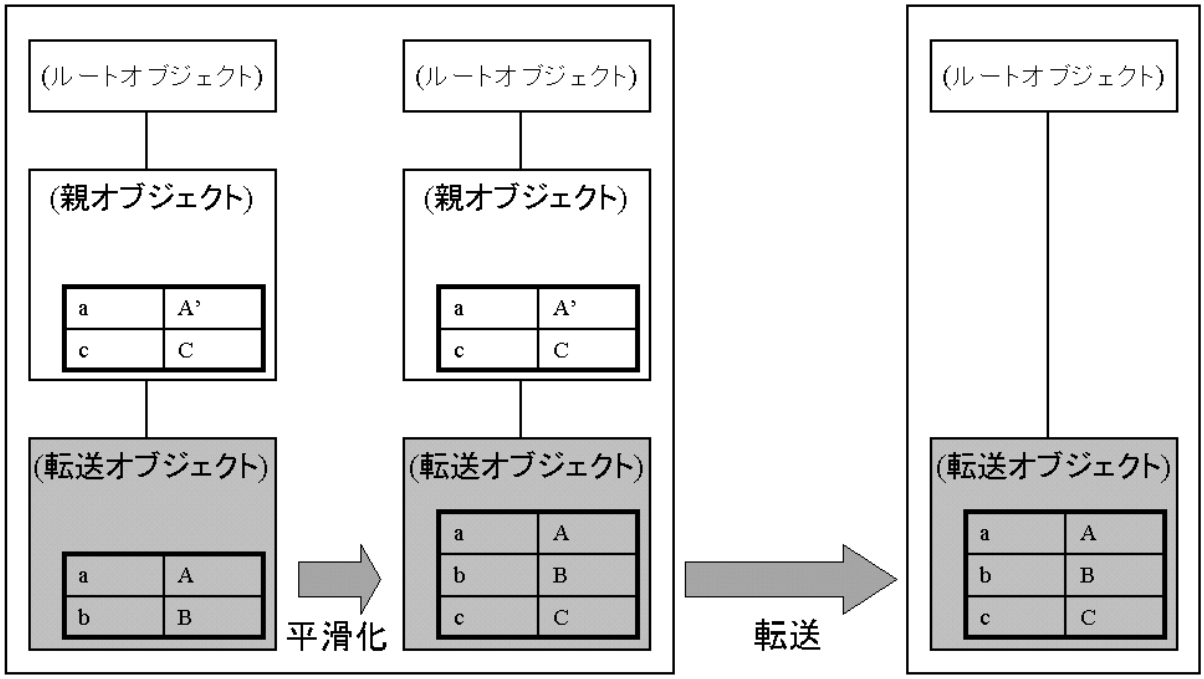


図 6.12: オブジェクトの平滑化

オブジェクトの後処理では、転送用に加工されたデータを戻す処理、画面に登録する処理、イベントリスナー[8]を設定する処理を行う。タートルオブジェクトと図形オブジェクトでは、転送用のデータから画像と幾何学データを復元する。ボタンなどの GUI オブジェクトとタートルオブジェクト、図形オブジェクトでは、画面に登録する処理が行われる。ボタンオブジェクトとテキストフィールドオブジェクトでは、マウスやキーボードからのイベントを受け取れるようにリスナーの設定を行う。これらの処理により、複製したオブジェクトがクライアント画面に表示され、適切な動作を行えるようになる。

複製したオブジェクトの親オブジェクトを設定する。親オブジェクトは、存在すればプロトタイプオブジェクトを設定し、そうでない場合はルートオブジェクトを設定する。この処理により、複製したオブジェクトをローカルで実行できるようになる。

複製処理はオブジェクトをローカル側で生成する処理までを行う。広域変数への登録は、複製の結果を変数に代入する形でプログラム中で明示的に行う必要がある。

サーバー側では、次の手順で複製を行う。

- 名前テーブル (objmap) からオブジェクトを検索する

[8]ボタンの押下など、システム的なイベントを受信するためのオブジェクト。

- 検索されたオブジェクトを返す

サーバー上のオブジェクトは、平滑化や直列化できないデータの変換など、登録時に転送に適した形式に加工されている。よって、複製時の前処理は不要である。

6.3.6 共有

カメ吉＝サーバー！”t1” 共有。

ドリトルのオブジェクト共有は、サーバー上のオブジェクトを複数のクライアントから共有する。基本的な考え方は、サーバー側のオブジェクト (リモートオブジェクト) の参照をクライアントに置き、それにメッセージを送り実行するモデルである。

実際の実装では、次の特性を考慮し、メッセージを中継するプロキシだけではなく、クライアント側にドリトルのオブジェクトを生成することで、参照と実体を混在させる形を取った。

- ドリトルのプログラムでは、画面に表示されるオブジェクトが使われることが多い。
- 画面上で他のオブジェクトとの衝突や、ボタン押下によるイベント処理などが必要になる

図 6.13 に、オブジェクト共有の構成を示す。クライアントの “サーバー” オブジェクトは、“共有” メッセージを受け取ると、次の手順でサーバー側のオブジェクトからローカルの共有オブジェクトを生成する。

- サーバーに getShared() メソッドを実行する
- サーバーから、リモートオブジェクトの参照が返る
- ローカルオブジェクトの核となるプロキシオブジェクト (OxRemote) を生成し、リモートオブジェクトへの参照 (sv) を格納する
- リモートオブジェクトに getClass() を実行し、リモートオブジェクトのクラスを調べる

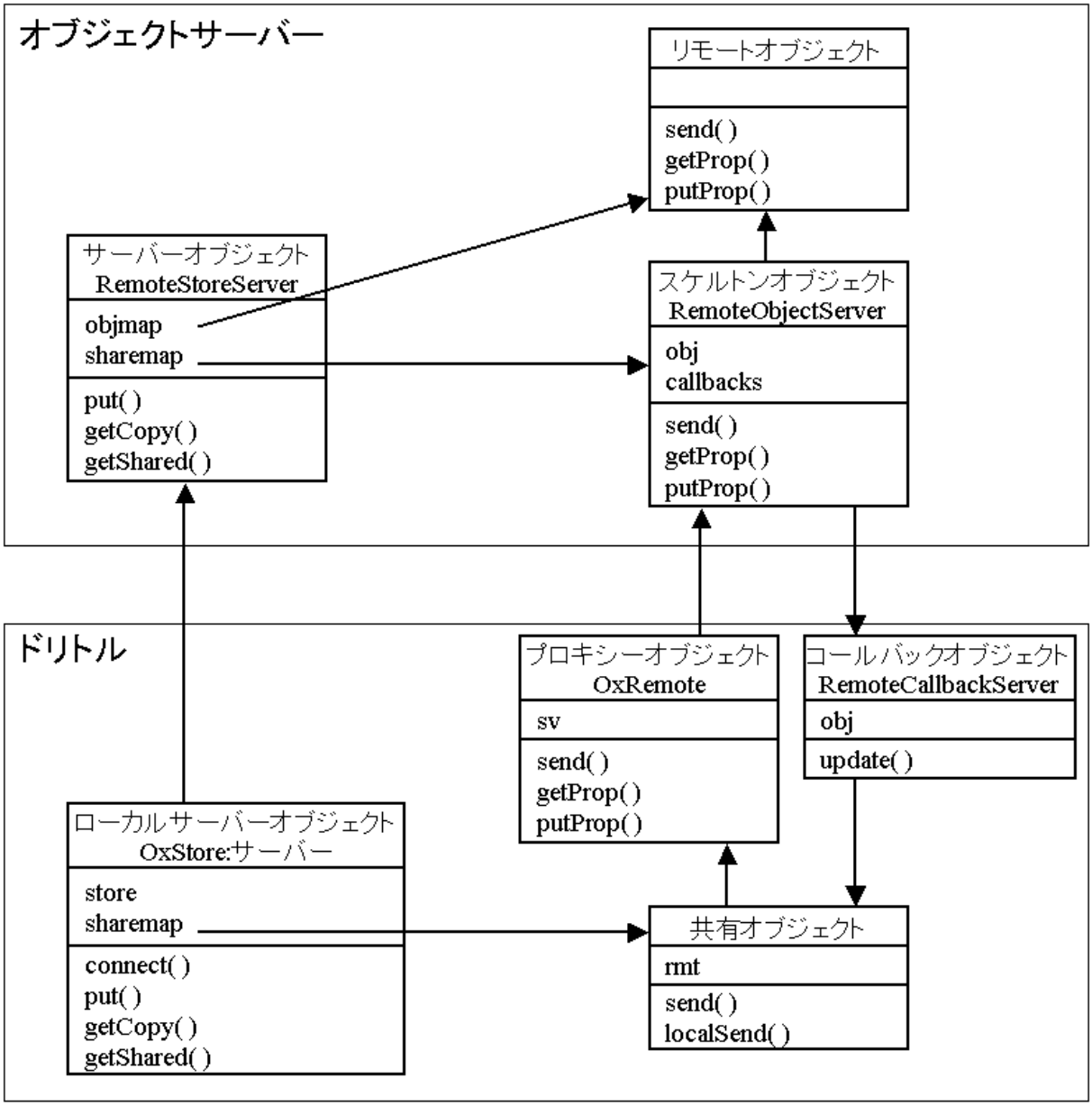


図 6.13: オブジェクトの共有

- ローカル側に、リモートオブジェクトと同じクラスのインスタンスを生成する。これが共有オブジェクトとなる
- 生成した共有オブジェクトへのメッセージが OxRemote オブジェクトに転送されるようにする. 以後、共有オブジェクトへのメッセージやプロパティ操作は OxRemote を通してサーバー側で実行される
- 共有オブジェクトのスケルトンであるコールバックオブジェクト (RemoteCallback-Server) を生成し、サーバーに伝える
- リモートオブジェクトの内部状態を共有オブジェクトに複製する
- 共有オブジェクトに複製後の後処理をする。後処理では、転送用に加工されたデータを戻す処理、画面に登録する処理、イベントリスナーを設定する処理を行う
- ローカルサーバーオブジェクト (OxStore) に、共有オブジェクトとコールバックオブジェクトを登録する

サーバー側では、次の手順で共有処理を行う。

- 名前テーブル (objmap) からオブジェクトを検索する
- 検索されたオブジェクトのスケルトンを作り、スケルトンテーブル (sharemap) に入れる
- スケルトンをクライアントに返す

また、クライアントが生成したコールバックオブジェクトは、スケルトン中のテーブル (callbacks) に格納される。

6.3.7 共有実行

カメ吉！100 歩く。

共有オブジェクトに送られたメッセージは、次の手順で実行される。

- クライアントの共有オブジェクトは、リモートオブジェクトのスケルトン (RemoteObjectServer) に send() メッセージを転送する
- リモートオブジェクトのスケルトンは、リモートオブジェクトを共有しているクライアントのコールバックオブジェクトに対して、「順に」update() メッセージを転送する[9]
- クライアント側のコールバックオブジェクトは、スケルトンからメッセージを受け取ると、共有オブジェクトの localSend() を実行し、クライアント側でのメソッド実行を行う
- リモートオブジェクトのスケルトンは、リモートオブジェクトに send() メッセージを送り、サーバー側での実行を行う
- 実行の結果がクライアントに返る

6.3.8 最適化

共有オブジェクトの実行は、当初は実行速度が遅く実用性に問題があった。そこで、2種類の最適化を行うことで改善を行った。

リモート呼び出しの最適化

当初はクライアント・サーバー間のすべての通信でオブジェクトを転送していたが、次の 2 つの問題があった。

- 実行速度の問題。タートルオブジェクトなど、画像のように大きなデータを持つオブジェクトを毎回転送するのはシステムへの負荷が高い
- カスケード実行の問題。実行の結果としてローカルオブジェクトが作られると、カスケード送信を行えない。次のプログラム例で、“100 歩く” はリモートオブジェク

[9]現在の実装では、メッセージはクライアントとサーバーでシリアルに実行される。あるオブジェクトを 2 つのクライアントから共有している場合には、メッセージの処理は「クライアント (× 2)+サーバー (× 1)」の時間が必要である。

ト (カメ吉) に送られるが、“90 右回り” は実行の戻り値であるローカルオブジェクトに送られてしまう。

カメ吉！100 歩く 90 右回り。

そこで、リモート実行の結果を参照で返すよう修正を行った。その結果、共有オブジェクトに対するメッセージのカスケード実行が可能になった。

リモートで実行された結果として返るオブジェクトは、共有オブジェクトとして各クライアントに生成される。クライアント側では、共有オブジェクトの参照テーブルを管理する。同じリモートオブジェクトに対応する共有オブジェクトは、テーブルから引いて返される。新たに共有オブジェクトを生成する処理は、画像などの転送が各クライアントに対して行われるため重い処理になるが、ドリトルのプログラムでは自分自身 (self) を返すメソッドが多いため、実際のプログラムにおいて大きな速度低下が発生することはなかった。

プロパティ参照の最適化

ドリトルでは、オブジェクトのプロパティにブロックを格納することでメソッドを定義する。また、タートルオブジェクトと図形オブジェクトは、画面上で位置を移動したときに他のオブジェクトと重なると、相手から “衝突” というメッセージを受け取る。

共有オブジェクトは衝突メッセージを受け取ると、サーバーにメッセージを転送する。サーバーはオブジェクトを共有しているクライアントにコールバックする形で、各クライアント上でメッセージを実行する。クライアントでの実行時には、メソッドの定義を取得するために、サーバーに対して衝突という名前のプロパティを参照する。この処理において、衝突プロパティの値であるブロックの転送に時間がかかるという問題があった。

- 画面でオブジェクト同士が重なるたびに衝突メソッドが実行される。(回数が多く頻繁)
- 共有されたクライアントの数だけブロックの転送が行われる
- ブロックの転送は重い処理である

そこで、ブロックが immutable なオブジェクトであることを利用し、キャッシュすることでパフォーマンスの向上を図った。改良したプロパティ値の参照手順を示す。

- サーバーにプロパティの型を問い合わせる
- ブロック以外なら、値を転送する
- ブロックであれば、値の代りにハッシュ値を転送する。ローカル側では、ブロックのキャッシュを参照し、ヒットしない場合のみブロックを転送する

ブロック以外の immutable なドリトルのオブジェクトには数値、文字列、色、論理値などが存在するが、これらはデータ量が小さく転送のコストが軽いため、今回のキャッシュの対象とはしなかった。

最適化の効果

リモート呼び出しの最適化とプロパティ参照の最適化の効果を測定した。使用したプログラムを図 6.14 に示す。このプログラムでは、サーバー上のタートルオブジェクトを共有し、そのオブジェクトに “歩く” メッセージを 200 回繰り返し送信する。結果を表 6.2 に示す。実行時間は、1 回あたりの時間である。測定に用いた環境は、クロック 750MHz の PentiumIII プロセッサを備えたパーソナルコンピュータを使用した。メモリは 384MB を搭載し、OS は Windows 2000 Professional である。リモートオブジェクトでの実行で、サーバーを localhost に置く測定では、このマシン上でドリトルとオブジェクトサーバーを同時に実行した。

```
// リモートオブジェクトでの実行
サーバー!"localhost"接続。
t=サーバー!"t"共有。
b=ボタン!"実行"作る 100 0 位置。
b:動作=「「t!1 歩く」!200 繰り返す」。
```

```
// ローカルオブジェクトでの実行
t=タートル!作る。
b=ボタン!"実行"作る 100 0 位置。
b:動作=「「t!1 歩く」!200 繰り返す」。
```

図 6.14: 共有実行速度測定プログラム

表 6.2: 共有実行速度の測定結果

テスト	実行時間 (msec)	説明
最適化前	520	リモートオブジェクト (サーバーは localhost で実行)
最適化後	23.5	リモートオブジェクト (サーバーは localhost で実行)
最適化後	26.0	リモートオブジェクト (サーバーは別ホストで実行)
ローカル	0.34	ローカルオブジェクト

リモートオブジェクトでの実行では、別ホストにサーバーを置く測定も行った。サーバーを置いた環境は、クロック 1.8GHz の Celeron プロセッサを備えたパーソナルコンピュータを使用した。メモリは 512MB を搭載し、OS は Windows 2000 Professional である。2 台のマシン環境は 100BASE-TX のネットワークで接続した。

ドリトルのプログラムでは、タイマーオブジェクトを利用した定期的な繰り返し処理が用いられることが多い。スムーズなアニメーション表示を行うためには 0.1 秒程度の間隔でメッセージを処理する必要がある。そこで、今回の測定では 0.1 秒以内で実行できることを評価の目安とした[10]。

最適化前の結果を見ると、1 回のメッセージ処理に約 0.5 秒を要している。メッセージを 0.1 秒以内に処理できないことから、共有オブジェクトを実用的なプログラムで使用することは困難であったことがわかる。

最適化後の結果を見ると、1 回のメッセージ処理に 0.0235 秒を要している。メッセージを 0.1 秒以内に処理できることから、実用的なプログラムで使用可能な速度に改善されたことがわかる。サーバーを別ホストに置いた場合でも、1 回のメッセージ処理は 0.026 秒であり、大きな速度の変化は見られなかった。

参考までに、共有オブジェクトの代りにローカルオブジェクトで実行した結果を測定した。ローカルオブジェクトと比較したリモートオブジェクトの速度比は約 70 倍であった。

[10]ドリトルに付属する 9 種類のサンプルプログラムを調べたところ、タイマーの繰り返し間隔は 0.1 秒以上のプログラムが多いことがわかった。ブロック崩しゲームなど、一部の速度が要求されるプログラムを含めても、タイマーの繰り返し間隔は最短で 0.05 秒であった。

6.4　授業での利用

分散共有ドリトルが生徒に理解可能であることを検証するために、2 種類の実験授業を行った。中学校で行った実験授業では、3 人の生徒を対象に 2 時間の授業を行い、中学生がオブジェクトの登録と複製を理解し、扱えることを確認した。企業で行った実験授業では、15 人の新入社員を対象に 3 日間の授業を行った。分散共有を含むプログラミングを体験することで、ネットワークの情報通信を含む計算機の原理を体験的に学ぶことができた。

これらの実験授業は、分散共有ドリトルの設計に大きな問題がないことを検証することを目的として行った。学校教育現場での分散共有ドリトルの教育に関する実験授業は、別の機会に行う予定である。

6.4.1　中学校での実験授業

実験概要

実験授業は公立中学校 (静岡県藤枝市立西益津中学校) で実施した。授業は放課後を利用し、1 時間の授業を一週間間隔で 2 回行った。授業は筆者が担当した。

授業の対象は 2 年生の生徒 3 人である。生徒は全員がワードプロセッサや電子メールを使用した経験があり、基本的なキーボードやマウスの操作は問題なく行えた。プログラミングについては、1 学期に 5 時間行われたドリトルの授業の中で、全員がタートルグラフィックス、メソッド、ボタンの概念を学んでいた。それ以前にプログラミングの経験のある生徒はいなかった。

実験方法は、分散共有ドリトルを生徒に講習し、生徒が実習を行った。授業中に生徒が学ぶ様子と、生徒の発言を観察した。

実験の進め方

表 6.3 に実施したカリキュラムを示す。

表 6.3: 実験授業のカリキュラム

授業	ねらい	内容
1	オブジェクトを転送するプログラミング	ネットワークの説明
		オブジェクトの登録
		オブジェクトの複製
2	オブジェクトを共有するプログラミング	数当てクイズプログラム
		石取りゲームプログラム

実験結果

説明の後、生徒には自作のオブジェクトを作ってもらい、各人のオブジェクトをサーバーに登録した。図 6.15 に生徒が作成したプログラムを示す。

全員が自作のオブジェクトをサーバーに登録した後、他の 2 人のオブジェクトを自分のドリトルに取り込んで実行した。図 6.16、図 6.17 に生徒のプログラム例を示す。このプログラムでは、サーバーに「カメゾウ」という名前のタートルオブジェクトを登録した後、他の生徒が登録したタートルオブジェクトとボタンオブジェクトを取り込んでいる。取り込んだオブジェクトがオブジェクトの性質を持つことを確認するために、タートルオブジェクトの「三角」メソッドを実行したときに画面に三角形が描かれることと、ボタンオブジェクトを押下したときにタートルの画像が変化することを確認した。

次に、石取りゲームを扱った。これは、「石の山から交互に数個ずつ石を取り合い、最後の 1 個を取ったほうが負け」というゲームである。今回は、最初の石の数を 10 個とし、同時に取れる石は 1〜3 個というルールとした。全員が図 6.18 のプログラムを実行し、石を取って戻す操作を順番に行うことにより、ゲームを行った。

続いて、図 6.18 のプログラムについて、ゲームの動作と照らし合わせながら生徒と読み合わせを行った。

このゲームではネットワーク上のひとつの資源 (石の数を表す変数) に複数のプログラムがアクセスを行うが、生徒から「石を取って戻したのに、数が正しく減らないことがあった。他の人と同時に操作したからだろうか」という質問があり、議論になった。例として、「サーバー上に 10 個の石が存在するとき、2 人の生徒が同時にその石を取り、それぞれ 2 個と 3 個減らしてサーバーに戻すと、サーバー上には何個の石が残るか」という問

```
// 生徒A
サーバー！"192.168.1.24" 接続。
カメゾウ＝タートル！作る 。
カメゾウ:星＝「「!100 歩く 144 右回り」!5回 繰り返す」。
サーバー！"カメゾウ星"（カメゾウ）登録。

// 生徒B
サーバー！"192.168.1.24" 接続。
カメ太＝タートル！作る "rocket.gif" 変身する。
カメ太:三角＝「「!100 歩く 120 右回り」!3回 繰り返す」。
サーバー！"カメ太三角"（カメ太）登録。

// 生徒C
サーバー！"192.168.1.24" 接続。
変身＝ボタン！"変身" 作る。
変身：動作＝「カメゾウ！"99.gif" 変身する」。
サーバー！"変身ボタン"（変身）登録。
```

図 6.15: 生徒の登録プログラム

図 6.16: 複製プログラムの実行例 (右図はボタン押下後)

```
サーバー！"192.168.1.24" 接続。
カメゾウ＝タートル！作る。
カメゾウ:星＝「「!100 歩く 144 右回り」!5回 繰り返す」。
サーバー！"カメゾウ星"（カメゾウ）登録。

カメ太＝サーバー！"カメ太三角" 複製。
カメ太！三角。
変身ボタン＝サーバー！"変身ボタン" 複製。
```

図 6.17: 生徒の複製プログラム例

```
// 画面に部品を並べる
もらう=ボタン！"もらう" 作る 0 100 位置。
とる=ボタン！"とる" 作る 0 50 位置。
表示=フィールド！作る 0 0 位置。
おくる=ボタン！"おくる" 作る 0 -50 位置。

// サーバーにつなぐ
サーバー名="192.168.1.24"。
サーバー！(サーバー名) 接続。

// サーバーに「山」を登録する。石の数を表す
山=10。
サーバー！"山"（山）登録。

// ゲームを作る
もらう:動作=「山=サーバー!"山" 複製。表示！(山) 書く」。
とる:動作=「山=(山 − 1)。表示！(山) 書く」。
おくる:動作=「サーバー!"山"（山）登録。表示！"" 書く」。
```

図 6.18: 石取りゲーム

いかけをしたところ、「5 個」という予想が出た。そこで生徒にプログラムの動作を口頭で発言してもらいながら、黒板で動作のシミュレーションを行い、8 個になる場合と 7 個になる場合があることを検証した。

このように、ネットワーク上でデータを共有する体験により、1 台の計算機だけのプログラミングでは気づかない問題を発見し、議論の中で深めていくことが可能であった。

6.4.2 企業での新入社員教育

実験概要

企業の研修でドリトルを使用し、プログラミングの授業を行った。コース名は「コンピュータの動作原理」である。授業は筆者が担当した。

対象となった生徒は、営業希望または SE 希望の新入社員 15 人である。プログラミング経験は、研究等で使用していた生徒が 2 名、大学の授業で体験した生徒が 2 名、経験のない生徒が 11 名という内訳であった。

実験方法は、分散共有ドリトルを生徒に講習し、生徒が実習を行った。授業中に生徒が学ぶ様子と、生徒の発言を観察した。

生徒は前日までの 2 日間でパーソナルコンピュータをキットから組み立てる講習を受講し、今回の 3 日間の授業に臨んだ。計算機の原理のうち、ハードウェアについては前日までに部品の構成を学んでいることから、ドリトルを使い計算機がソフトウェアで動いているという側面を体験的に学ぶことを授業の中心とした。

実験の進め方

表 6.4 にカリキュラムを、図 6.19 に授業風景を示す。1 日目の午前は、計算機の構成を説明した後、ドリトルの動作に必要なソフトウェアとして、Java とドリトルのインストールを行った。1 日目の午後は、プログラミングの体験として、タートルグラフィックスを扱った。オブジェクトに命令を送ると実行が行われるという基本的な考え方を、画面上のオブジェクトの移動を確認しながら学習した。2 日目の午前は、タイマーによるアニメーションを題材に、繰り返しとスレッドの概念を学んだ。2 日目の午後は、ボタンが押されたときの動作とタートルが他のオブジェクトと衝突したことの検知を題材に、オブジェク

トのメソッド定義を扱った。続いて、教室内の LAN 環境と各端末がネットワークで接続されていることを説明し、以下の分散共有ドリトルの機能を扱った。

- オブジェクトの転送。押すと何らかの動作を行うボタンを作り、サーバーに登録する。続いて、他の生徒のボタンをサーバーから複製し、端末上で実行する
- オブジェクトの共有。サーバー上のタートルオブジェクトを全端末から共有し、生徒が作ったプログラムから一斉に操作する

3 日目の午前は、各自の作品プログラムを作成する時間とした。3 日目の午後は、発表の準備として PowerPoint 等でスライドを作った後、発表会を行った。

表 6.4: 新人研修のカリキュラム

時間	内容
1 日目 (AM)	計算機とソフトウェアの関係。Java とドリトルのインストール
1 日目 (PM)	はじめてのプログラミング
2 日目 (AM)	タイマーによる繰り返し
2 日目 (PM)	メソッド定義、ネットワーク
3 日目 (AM)	作品製作
3 日目 (PM)	作品発表会

実験結果

表 6.5, 表 6.6 に授業の前後に行ったアンケートの結果を示す。以下は、結果から読み取れる内容である。

- (1) 全員が情報通信を支えるソフトウェアの存在を理解した (強い肯定が 67%から 100%に増加)
- (2) 楽しさを感じ (肯定が 40%から 93%に増加)、(4) 難しいという印象が減少した (肯定が 80%から 46.7%に減少)

図 6.19: 新人研修の授業風景

- (5) プログラムは論理で書かれており、正しく書けば動く (強い肯定が 67%から 93%に増加) が、(3) きちんと書かないと動かない (強い肯定が 67%から 80%に増加)

(1) は情報通信の中で使われるソフトウェアの存在を問う設問である。授業前は弱い肯定が見受けられたが、授業後は全員が強い肯定に変化したことから、分散共有ドリトルによるプログラミングを体験することで、「WWW や電子メールといった情報通信がソフトウェア技術に支えられている」ことを体験的に学べることを確認した。

6.5 まとめ

本章では、ドリトルを拡張し、ネットワーク上でオブジェクトを活用することが可能な分散共有ドリトルを提案した。

分散共有ドリトルを使うことで、たとえばタートルオブジェクトを共有して操作しながら、オブジェクトがネットワークを介して動作する様子を視覚的に観察する授業が可能になる。このような、ひとつの実体 (Model) を複数の視点 (View) から参照するモデルを、Observer パターンのような特別なコードを記述せずに扱う機能を提供した。

表 6.5: 理解度アンケートの結果 (授業前)

設問		肯定 ←			→ 否定	平均
		4	**3**	**2**	**1**	
(1)	WWW や携帯電話の裏ではプログラムが動いている	66.7%	26.7%	6.7%	0.0%	3.60
(2)	プログラミングは楽しい (楽しそう)	6.7%	33.3%	33.3%	26.7%	2.20
(3)	プログラムはきちんと書かないと動かない	66.7%	26.7%	6.7%	0.0%	3.60
(4)	プログラミングは難しい (難しそう)	40.0%	40.0%	20.0%	0.0%	3.20
(5)	プログラムは正しく書けばそのとおりに動く	66.7%	26.7%	6.7%	0.0%	3.60

表 6.6: 理解度アンケートの結果 (授業後)

設問		肯定 ←			→ 否定	平均
		4	**3**	**2**	**1**	
(1)	WWW や携帯電話の裏ではプログラムが動いている	100.0%	0.0%	0.0%	0.0%	4.00
(2)	プログラミングは楽しい (楽しそう)	26.7%	66.7%	6.7%	0.0%	3.20
(3)	プログラムはきちんと書かないと動かない	80.0%	13.3%	6.7%	0.0%	3.73
(4)	プログラミングは難しい (難しそう)	13.3%	33.3%	46.7%	6.7%	2.53
(5)	プログラムは正しく書けばそのとおりに動く	93.3%	6.7%	0.0%	0.0%	3.93

続いて、分散共有ドリトルを用いた2つの実験授業を行った。中学校での実験授業では2時間の授業を行い、3人の生徒がオブジェクトの登録・転送機能を理解し、扱うことが可能であることを確認した。また、サーバー上のデータを同時に更新するプログラムを扱うことを通して、ネットワーク上で動作するプログラムがローカルに動作するプログラムとは異なるという側面を体験することができた。企業での実験授業では3日間の授業を行い、15人の新入社員がドリトルを利用したプログラミングを体験した。授業の中でオブジェクトの登録・複製・共有を含む分散共有の機能を扱うことで、計算機の単体の動作原理に加えて、計算機同士がネットワークを介して通信し合う様子を体験的に理解することができた。

以下は、分散共有ドリトルで今後検討すべき課題である。

- 授業カリキュラムの作成。今回の実験授業では数人の中学生を対象とした授業を行った。今後、学校教育の授業に取り入れて活用するためには、分散共有についてのカリキュラムとテキストの整備が課題である。

- 高速化。本章ではブロック転送の最適化について述べたが、画像など転送にコストがかかるデータについて、改善の余地が残されている。また、現在はコールバック処理をシリアルに行っているため、メッセージの実行時間はクライアント数に比例して遅くなる。コールバック処理を非同期に行うことで速度の改善が期待できる。

- 実用的なプログラムの記述。学習用の教材として実用的なシステムを構築する場合には、オブジェクトの排他制御やトランザクション管理、メッセージの順序実行の保証などが必要になると考えられる。

- 実行結果の複製による返戻。現在は、共有オブジェクトへのメッセージ送信の結果はメソッドが返したオブジェクトが共有可能であれば常に共有オブジェクトとして返される。この仕様はカスケード実行の実現と実行速度向上の上で効果があったが、共有を望まない場合でも共有が強制されてしまうことになる。この部分を制御可能にすることは、今後の検討課題である。

今後は、これらの課題への対応を行いながら、情報教育での活用を進めていきたい。

第7章　結論

本論文では、学校教育で利用されるプログラミング言語に求められる性質と条件についての考察を行い、その考えに基づいた新しいオブジェクト指向言語「ドリトル」を提案し、実際の授業において評価を行った。また、ドリトルを分散環境に拡張した分散共有ドリトルを提案し、設計と実装について解説した。

第1章では、学校教育で行われている情報教育を概観し、情報教育で利用するためにプログラミング言語に求められる性質を考察した。

第2章では、情報教育で使われることのある各種のプログラミング言語について述べた。C++[35] や Java[13] などオブジェクト指向を取り入れた汎用言語は、習得が難しいことから学校教育の中で用いられることはほとんどない。また、BASIC や LOGO など現在までに提案されている入門・教育用の言語や、図や例示によってプログラミングを行う簡易的な言語についても考察し、「構文が単純で理解しやすい」性質だけでは不十分であり、画面上の GUI 部品などのオブジェクトを操作することで生徒が日常接するアプリケーションソフトウェアを類推できることが必要であり、ネットワークを扱える言語が望ましいことを示した。

第3章では、学校教育に適したプログラミング言語である「ドリトル」を提案した。ドリトルは、今日のソフトウェア技術において主要な位置付けを占めるオブジェクト指向 [24] の考えに基づいた教育用プログラミング言語である。従来、教育現場で使われていた BASIC[17] や LOGO[26] などの言語が、フローチャートに代表される「計算機の考え方」に沿ってプログラムを記述するのに対し、ドリトルは動物やボタンなど実世界の「もの」に相当する「オブジェクト」を単位としてプログラムを記述することで、比較的人間に近い考え方でプログラムを記述できる。オブジェクト指向言語の習得は難易度が高いとされるが、ドリトルでは、オブジェクトを複製によって生成するプロトタイプ方式のオブジェクト指向を採用することで、初心者に理解が難しいクラスの概念を使わずにオブジェクトを扱えるようにした。また、構文は日本語の変数名や命令語を用いることで英語の壁

をなくし、プログラムに用いる記号を少なくする構文設計を行うことで、習得を容易にした。

第 4 章では、小規模な 2 種類の実験授業により、設計したドリトルの言語仕様を確認した。高等学校で行った実験授業では、3 人の高校生に 3 時間の授業を行った。数時間の授業で生徒が学んだ内容は、生徒が作成したプログラムを分析することで行った。その結果、タイマーによるスレッドや、プロパティ階層を用いた継承など、高度な概念を理解し、プログラムとして活用していることを確認した。また、プログラミングを経験したことのない生徒の学習を観察した。生徒の発言やアンケートの感想から、プログラミングを体験することで、画素の存在やオペレーティングシステムの存在など、さまざまな計算機の原理を発見しながら学習できることを確認した。教員を対象にした実験授業では、小学校から大学までの幅広い教員 20 人に 1.5 時間の授業を行った。教員の立場から意見をもらうことで、学校教育の授業で使用した場合に大きな問題がないことを確認した。実施した 2 種類の実験授業を通して、設計したドリトルの言語仕様の妥当性を確認することができた。

第 5 章では、実際の学校教育の現場で実験授業を行った。授業は、公立中学校において技術科の正規の授業として実施し、132 名の生徒が受講した。授業は担当教員が行い、筆者は、生徒の理解度や授業の様子を評価するために、2 種類のアンケートと 2 回の期末試験の問題を作成した。これらの結果と生徒が作成したプログラムを分析した結果、中学校における 10 時間程度の授業において、生徒の興味が最後まで持続し、適度な難易度を持つ授業を構築できることを確認した。授業が進むにつれて、学習内容の難易度は高くなるが、生徒の楽しさは増していった。これはプログラミングという題材の力であると同時に、ドリトルがプログラミングの魅力をわかりやすい形で生徒に提示できた結果であると考えている。生徒たちは、最後の数時間で GUI 部品 (特にボタン) を活用し、簡単なペイントソフトを作成した。日常使っているアプリケーションソフトウェアに結び付くプログラミングを体験したことで、生徒たちは大きな達成感を得ることができた。一般の言語において、GUI 部品を利用したプログラムの作成は容易でないことが多いが、ドリトルでは、プログラミングを学び始めて数時間の生徒がボタンを使ったプログラムを作成することができた。

第 6 章では、ドリトルを拡張し、ネットワーク上でオブジェクトを活用することが可能な分散共有ドリトルを提案した。分散共有ドリトルを使うことで、たとえばタートルオブジェクトを共有して操作しながら、オブジェクトがネットワークを介して動作する様子

を視覚的に観察する授業が可能になる。このような、ひとつの実体 (Model) を複数の視点 (View) から参照するモデルを、Observer パターンのような特別なコードを記述せずに扱う機能を提供した。続いて、分散共有ドリトルを用いた 2 つの実験授業を行った。中学校での実験授業では 2 時間の授業を行い、3 人の生徒がオブジェクトの登録・転送機能を理解し、扱うことが可能であることを確認した。また、サーバー上のデータを同時に更新するプログラムを扱うことを通して、ネットワーク上で動作するプログラムがローカルに動作するプログラムとは異なるという側面を体験することができた。企業での実験授業では 3 日間の授業を行い、15 人の新入社員がドリトルを利用したプログラミングを体験した。授業の中でオブジェクトの登録・複製・共有を含む分散共有の機能を扱うことで、計算機の単体の動作原理に加えて、計算機同士がネットワークを介して通信し合う様子を体験的に理解することができた。

本論文では、学校教育において求められるプログラミング言語の条件を明らかにし、学校教育で利用可能な新しいオブジェクト指向言語「ドリトル」を提案した。設計・実装したドリトルを小規模な実験授業で評価することで仕様の確認を行い、続いて実際の学校教育で実験授業を行い、生徒が高い意識で取り組む形で授業が成立することと、実際に学習可能な言語であることを実証した。さらに、ドリトルを分散に拡張した分散共有ドリトルを提案し、実装の詳細とネットワークへの応用の可能性を論じた。本論文で提案した言語「ドリトル」は実用性が高く、小学校から高等学校までの学校教育で活用することが可能である。今後は多くの学校で使ってもらいながら、教材や機能をさらに充実させていきたい。

参考文献

[1] Harold Abelson and Amanda Abelson. *Logo for the Macintosh: An Introduction Through Object Logo*. Paradigm Software, 1992.

[2] Dave Baum, Michael Gasperi, Ralph Hempel, and Luis Villa. *Extreme MINDSTORMS : an advanced guide to LEGO MINDSTORMS*. Technology in action series. Apress, 2000.

[3] Alfred Bork. *Personal Computers for Education*. Harper & Row, 1985. (邦訳:塚本栄一. 21 世紀に向けた学校教育とコンピュータ. 丸善, 1991).

[4] Allen Cypher and David Canfield Smith. KidSim: End user programming of simulations. In *Proceedings of the ACM conference on Human factors in computing systems(CHI'95)*, pp. 27–34, 1995.

[5] Andrea diSessa and Harold Abelson. Boxer: a reconstructible computational medium. *Communications of the ACM*, Vol. 29, No. 9, pp. 859–868, 1986.

[6] European Computer Manufacturers Association. ECMAScript language specification, 1999. http://www.ecma-international.org/publications/standards/Ecma-262.htm.

[7] E.W.Dijkstra. Go to statement considered harmful. *Communications of the ACM*, Vol. 11, No. 3, pp. 147–148, 1968.

[8] Etienne Gagnon. SableCC, an object-oriented compiler framework. Master's thesis, McGill University, 1998. http://www.sablecc.org/.

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[10] G.Krasner and S.T.Pope. A cookbook for the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming*, Vol. 1, No. 3, 1988.

[11] Richard G.McDaniel and Brad A.Myers. Getting more out of programming-by-demonstration. In *Proceedings of the ACM conference on Human factors in computing systems(CHI'99)*, pp. 442–449, 1999.

[12] Adele Goldberg and David Robson. *Smalltalk-80: The Language and Its Implementation*. Addison-Wesley, 1983.

[13] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[14] Satoshi Hirano. HORB: Distributed execution of java programs. In *Worldwide Computing and Its Applications*, No. 1274 in Lecture Note in Computer Science, pp. 29–42. Springer-Verlag, 1997.

[15] Martin Hitz and Marcus Hudec. Modula-2 versus C++ as a first programming language–some empirical results. In *Papers of the 26th SIGCSE technical symposium on Computer science education*, pp. 317–321, 1995.

[16] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. Back to the future: the story of Squeak, a practical Smalltalk written in itself. In *Proceedings of the 1997 ACM SIGPLAN conference on Object-oriented programming systems, languages and applications*, pp. 318–326, 1997.

[17] John G. Kemeny and Thomas E. Kurtz. *Basic programming*. John Wiley & Sons, 1967.

[18] Brian W. Kernighan and Dennis M. Ritchie. *The C programming language*. Prentice Hall, 1978.

[19] Michael Kölling, Bett Koch, and John Rosenberg. Requirements for a first year object-oriented teaching language. In *Papers of the 26th SIGCSE technical symposium on Computer science education*, pp. 173–177, 1995.

[20] Michael Kölling and John Rosenberg. Blue — a language for teaching object-oriented programming. In *Proceedings of the twenty-seventh SIGCSE technical symposium on Computer science education*, pp. 190–194, 1996.

[21] Henry Lieberman. Using prototypical objects to implement shared behavior in object-oriented systems. In *OOPSLA'87*, pp. 214–223, 1986.

[22] Mark Lutz. *Programming Python.* A nutshell handbook. O'Reilly, 1996.

[23] Richard G. McDaniel and Brad A. Myers. Getting more out of programming-by-demonstration. In *Proceedings of the ACM conference on Human factors in computing systems(CHI'99)*, pp. 442–449, 1999.

[24] Bertrand Meyer. *Object-oriented Software Construction.* Prentice Hall, 1994.

[25] Object Management Group. CORBA/IIOP 3.0 specification, 2002. http://www.omg.org/technology/documents/formal/corba_iiop.htm.

[26] Seymour Papert. *Mindstorms : children, computers, and powerful ideas.* Basic Books, 1980.

[27] Alexander Repenning and Corrina Perrone. Programming by analogous examples. *Communications of the ACM*, Vol. 43, No. 3, pp. 90–97, 2000.

[28] David Canfield Smith, Allen Cypher, and Kurt Schmucker. Making programming easier for children. *interactions*, Vol. 3, No. 5, pp. 58–67, 1996.

[29] David Canfield Smith, Allen Cypher, and Larry Tesler. Novice programming comes of age. *Communications of the ACM*, Vol. 43, No. 3, pp. 75–81, 2000.

[30] Randall B. Smith, Mark Lentczner, Walter R. Smith, Antero Taivalsaari, and David Ungar. Prototype-based languages (panel): object lessons from class-free program-

ming. In *Proceedings of the ninth annual conference on Object-oriented programming systems, language, and applications*, pp. 102–112, 1994.

[31] Randall B. Smith, John Maloney, and David Ungar. The Self-4.0 user interface: manifesting a system-wide vision of concreteness, uniformity, and flexibility. In *Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications*, pp. 47–60, 1995.

[32] Walter R. Smith. Using a prototype-based language for user interface: the Newton project's experience. In *Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications*, pp. 61–72, 1995.

[33] Guy L., Jr. Steele. *Common Lisp: The Language*. Butterworth-Heinemann, 1990.

[34] John Steinmetz. Computers and squeak as environments for learning. In *Squeak: Open Personal Computing and Multimedia*. Prentice Hall, 2001.

[35] Bjarne Stroustrup. *The C++ programming language*. Addison-Wesley, 1986.

[36] Sun Microsystems. Jini Network Technology. http://www.sun.com/software/jini/.

[37] David Ungar and Randall B. Smith. Self: The power of simplicity. In *OOPSLA'87*, pp. 227–242, 1987.

[38] Robert Ward and Martin Smith. JavaScript as a first programming language for multimedia students. In *Proceedings of the 6th annual conference on the teaching of computing/3rd annual conference on integrating technology into computer science education on Changing the delivery of computer science education*, pp. 249–253, 1998.

[39] 青山幹雄. コンポーネントウェア:部品組み立て型ソフトウェア開発技術. 情報処理学会誌, Vol. 37, No. 1, pp. 71–79, 1995.

[40] 市川伸一. コンピュータを教育に活かす. 勁草書房, 1994.

[41] 井戸坂幸男, 紅林秀治. 「ドリトル」ではじめる情報教育 第3回プログラミングっておもしろい〜中学校 技術・家庭科での授業実践. NEW 教育とコンピュータ, Vol. 18, No. 3, pp. 84–87, 2002.

[42] 上田哲郎, 久野靖. ベクターコンポーネント: コンポーネント結合による差分プログラミング. 情報処理学会論文誌, Vol. 40, No. SIG7(PRO4), pp. 40–50, 1999.

[43] 大岩元, 橘孝博, 半田亨, 久野靖, 辰己丈夫. 情報科教育法. オーム社, 2001.

[44] 大島芳樹, 脇田建, 佐々政孝. プログラミング言語処理系 Squeak の SHARP Zaurus への移植とその評価. 情報処理学会論文誌, Vol. 41, No. SIG9(PRO8), pp. 62–77, 2000.

[45] 河込和宏, 中村秀男, 大野邦夫, 飯島正. 分散オブジェクトコンピューティング. 共立出版, 1997.

[46] 片桐明. 日本語プログラミング言語 Mind. 翔泳社, 1988.

[47] 兼宗進, 中谷多哉子, 井戸坂幸男, 御手洗理英, 福井眞吾, 久野靖. 教育用オブジェクト指向言語「ドリトル」による授業実施とその評価. 情報処理学会 情報教育シンポジウム (SSS2002), pp. 229–236, 2002.

[48] 吉良智樹, 並木美太郎, 岩崎英哉. 初心者入門用言語「若葉」の言語仕様と処理系の実装. 情報処理学会論文誌, Vol. 40, No. SIG10(PRO5), pp. 28–38, 1999.

[49] 紅林秀治. 「ドリトル」ではじめる情報教育 第 4 回プログラミングっておもしろい〜ドリトルで制御の学習 その 1. NEW 教育とコンピュータ, Vol. 18, No. 4, pp. 120–123, 2002.

[50] 紅林秀治. 「ドリトル」ではじめる情報教育 第 5 回プログラミングっておもしろい〜ドリトルで制御の学習 その 2. NEW 教育とコンピュータ, Vol. 18, No. 5, pp. 40–43, 2002.

[51] 黒川利明. プログラミング言語の仕組み. 朝倉書店, 1997.

[52] 黄寧, 丸山桃代, 宮寺庸造, 横山節雄. プログラム可視化によるプログラミング教育支援. Technical Report ET99-1, 信学技報, 1999.

[53] 小谷善行. パソコン LOGO プログラミング. 東海大学出版会, 1984.

[54] 正田良. Logo で知る認知科学. 東京電機大学, 1999.

[55] 白石和夫. Windows95 で動作する Full BASIC 言語処理系. 情報教育シンポジウム論文集, pp. 135–140. 情報処理学会, 1999.

[56] 白石和夫. Windows 環境における JIS Full BASIC の実装. 情報処理学会論文誌, Vol. 41, No. SIG9(PRO8), pp. 52–61, 2000.

[57] 情報処理学会. エンサイクロペディア情報処理. オーム社, 2002.

[58] 鈴木勢津子. 考える力をはぐくむコンピュータ教育. 啓学出版, 1989.

[59] 関将俊. dRuby による分散オブジェクトプログラミング. アスキー, 2001.

[60] 総務省. 平成 14 年通信利用動向調査の結果, 2003. http://www.soumu.go.jp/s-news/2003/030307_1.html.

[61] 竹内郁雄. 初めての人のための LISP. サイエンス社, 1986.

[62] 竹内郁雄. ビジュアル言語はまだまだなっとらん. *bit*, Vol. 30, No. 1, pp. 20–21, 1998.

[63] 玉川理英. 入門 Visual Basic 6.0. きんのくわがた社, 1999.

[64] 中谷多哉子, 兼宗進, 御手洗理英, 福井眞吾, 久野靖. オブジェクトストーム: プログラミング言語と教育の新しい関係. 情報処理学会 OO2001 シンポジウム オブジェクト指向最前線, pp. 57–64. 近代科学社, 2001.

[65] 中村祐治, 市川道和, 大里治泰. 中学校 情報教育 Q&A. 教育出版, 1999.

[66] 日本規格協会. 電子計算機プログラム言語 Full BASIC (JIS X 3003:1993). 1993.

[67] 萩谷昌己. ソフトウェア考現学. CQ 出版社, 1985.

[68] 萩谷昌己. 視覚的プログラミングと自動プログラミング. コンピュータソフトウェア, Vol. 8, No. 2, pp. 27–39, 1991.

[69] 長谷川裕行. ソフトウェアの 20 世紀. 翔泳社, 2000.

[70] 馬場祐人. 日本語プログラミング言語 TTSneo. http://hp.vector.co.jp/authors/VA021321/.

[71] 増井俊之. ビジュアル言語のすすめ. *bit*, Vol. 30, No. 1, pp. 17–19, 1998.

[72] まつもとゆきひろ, 石塚圭樹. オブジェクト指向スクリプト言語 Ruby. アスキー, 1999.

[73] 三浦元喜, 田中二郎. Java アプレットのためのアニメーションヘルプシステム. 情報処理学会論文誌, Vol. 41, No. SIG4(PRO7), pp. 56–64, 2000.

[74] 文部科学省 情報化の進展に対応した初等中等教育における情報教育の推進などに関する調査研究協力者会議. 情報化の進展に対応した教育環境の実現に向けて 最終報告, 1998. http://www.mext.go.jp/b_menu/shingi/chousa/shotou/002/toushin/980801.htm.

[75] 文部省（編）. 高等学校学習指導要領. 1999.

[76] 文部省（編）. 小学校学習指導要領. 1999.

[77] 文部省（編）. 中学校学習指導要領. 1999.

[78] 文部科学省教育課程審議会. 幼稚園、小学校、中学校、高等学校、盲学校、聾学校及び養護学校の教育課程の基準の改善について (答申), 1998. http://www.mext.go.jp/b_menu/shingi/12/kyouiku/toushin/980703.htm.

[79] 山下淳, 葛岡英明, 鈴木栄幸, 加藤浩. 共同学習を支援するためのツール「アルゴカード」の開発. 研究報告「グループウェア」 23, 情報処理学会, 1997.

[80] 山本峰章. 日本語プログラム言語ひまわり. http://hima.chu.jp/.

謝辞

本論文をまとめるにあたり、入学以来御指導をいただいている筑波大学 久野靖教授に御助言と御指導をいただきました。また、筑波大学 鈴木久敏教授、津田和彦助教授、八重倉孝講師、椿広計教授、大木敦雄講師には、御教示・御助言をいただきました。感謝いたします。

本論文中の第 3 章の研究は、情報処理振興事業協会 (IPA) の平成 12 年度未踏ソフトウェア創造事業の補助を受けて行われ、電気通信大学 竹内郁雄教授から御指導をいただきました。感謝いたします。

第 4,5 章の研究の一部は、コンピュータ教育開発センター (CEC) の平成 13 年度 E スクエアプロジェクトの補助を受けて行われました。大日本図書株式会社 (当時) 原久太郎氏をはじめとするプロジェクトメンバーの皆様に感謝いたします。

第 3 章から第 6 章の研究は、共同研究者である株式会社アーマット 御手洗理英氏、有限会社エス・ラグーン 中谷多哉子氏、日本電気株式会社 福井眞吾氏に討論、御助言をいただきました。また、愛知教育大学 鎌田敏之助教授、金城学院大学 長谷川元洋助教授、高麗大学 李元揆教授には、大学教育や教員養成の視点から有益なコメントをいただきました。学校における実験授業の実施にあたっては、筑波大学附属高等学校 矢野一幸先生、兵庫県教育工学研究会 Logo 部会の皆様、松阪市立鎌田中学校 井戸坂幸男先生、藤枝市立西益津中学校 紅林秀治先生、鳥取県教育センター 足利裕人先生、鳥取県立青谷高等学校 足谷保典先生、千葉市立金沢小学校 佐藤和浩先生に多大なご協力をいただきました。授業の許可をいただいた各学校関係者の方々に感謝いたします。企業における新人教育の実施にあたっては、株式会社リコー 太田純氏に多大なご協力をいただきました。教育の機会をいただいた平岡昭夫室長、藤健児課長、斎藤信男課長に感謝いたします。

本研究は、上に記した方々をはじめ、多方面からの御指導と御協力のもとに完成したものであり、ここに謹んで感謝の意を表します。

教育マイクロワールド研究会会長の鈴木勢津子先生 (故人) には、学校教育における情報教育について多くの示唆をいただきました。先生は本論文が完成する直前、2003 年の秋に急逝された。心より御冥福をお祈りします。

最後に、平日は会社に、夜間と休日は研究に、という生活を続けながら本論文を完成できたのは、家族や友人たちのサポートのおかげである。心から感謝する。

本論文に関連した発表

査読付き学術論文

1. 兼宗進, 御手洗理英, 中谷多哉子, 福井眞吾, 久野靖. 学校教育用オブジェクト指向言語「ドリトル」の設計と実装. 情報処理学会論文誌, Vol. 42, No. SIG11(PRO12), pp. 78–90, 2001.

2. 中谷多哉子, 兼宗進, 御手洗理英, 福井眞吾, 久野靖. オブジェクトストーム: オブジェクト指向言語による初中等プログラミング教育の提案. 情報処理学会論文誌, Vol. 43, No. 6, pp. 1610–1624, 2002.

3. 兼宗進, 中谷多哉子, 御手洗理英, 福井眞吾, 久野靖. 初中等教育におけるオブジェクト指向プログラミングの実践と評価. 情報処理学会論文誌, Vol. 44, No. SIG13(PRO18), pp. 58–71, 2003.

予稿集他

1. 兼宗進, 久野靖. 学校教育用オブジェクト指向言語/環境の構想について. 情報教育シンポジウム (SSS2000), pp. 79–82, 2000.

2. 兼宗進, 久野靖. 学校教育用オブジェクト指向言語 “Dolittle” の提案. 第 42 回プログラミングシンポジウム, pp. 11–20, 2001.

3. 兼宗進, 中谷多哉子, 御手洗理英, 福井眞吾, 久野靖. オブジェクト指向言語「ドリトル」を利用した情報教育について. 情報教育シンポジウム (SSS2001), pp. 275–282, 2001.

4. 中谷多哉子, 兼宗進, 御手洗理英, 福井眞吾, 久野靖. オブジェクトストーム: プログラミング言語と教育の新しい関係. オブジェクト指向シンポジウム (OO2001), pp. 57–64, 2001.

5. 兼宗進, 中谷多哉子, 井戸坂幸男, 御手洗理英, 福井眞吾, 久野靖. 教育用オブジェクト指向言語「ドリトル」による授業実施とその評価. 情報教育シンポジウム (SSS2002), pp. 229–236, 2002.

6. 紅林秀治, 兼宗進, 岡田雅美, 佐藤和浩, 久野靖. 画面を飛び出したオブジェクト: 自立型ロボットを活用した情報教育の提案. 情報教育シンポジウム (SSS2002), pp. 77–84, 2002.

7. 兼宗進, 中谷多哉子, 御手洗理英, 福井眞吾, 久野靖. 教育用オブジェクト指向言語「ドリトル」を活用した 学校教育. 第 44 回プログラミングシンポジウム, pp. 13–24, 2003.

8. 兼宗進, 中谷多哉子, 御手洗理英, 福井眞吾, 久野靖. 端末を飛び出したオブジェクト: 分散プログラミングを活用した情報教育の提案. 情報教育シンポジウム (SSS2003), pp. 91–98, 2003.

学会発表

1. 兼宗進, 御手洗理英, 中谷多哉子, 福井眞吾, 久野靖. 学校教育用オブジェクト指向言語「ドリトル」の設計と実装. 情報処理学会 プログラミング研究会, 2001.

2. 兼宗進, 中谷多哉子, 御手洗理英, 福井眞吾, 久野靖. 初中等教育におけるオブジェクト指向プログラミングの実践と評価. 情報処理学会 プログラミング研究会, 2003.

3. 兼宗進, 中谷多哉子, 御手洗理英, 福井眞吾, 久野靖. 教育用プログラミング言語におけるオブジェクト共有機能の導入. 分散オブジェクト指向シンポジウム (SWoPP2003), 2003.

解説記事

1. 兼宗進. 「ドリトル」ではじめる情報教育 第1回 コンピュータっておもしろい! を体験しよう. NEW 教育とコンピュータ, Vol.18, No.1, pp.88–91, 2002.

2. 兼宗進. 「ドリトル」ではじめる情報教育 第2回 プログラミングで楽しもう. NEW 教育とコンピュータ, Vol.18, No.2, pp.84–87, 2002.

3. 兼宗進.「ドリトル」ではじめる情報教育最終回 〜こんな授業ができました. NEW 教育とコンピュータ, Vol.18, No.9, pp.108–111, 2002.

4. 兼宗進. 「ドリトル」のプログラミングで学ぶ情報教育. Educare, Vol.2002, No.3, pp.12–13, 2002.