平成 29 年度　卒業論文

*Scratckly:*プログラミング初学者のための ブロック型学習法

平成 30 年 2 月 15 日

14111040

三重野 剛

指導教員 三浦 元喜 准教授

九州工業大学 工学部 総合システム工学科

概 要

近年、ビジュアルプログラミング環境による初学者に対するプログラミング教育に注目が集まっている。初学者はビジュアルプログラミング環境を利用することで、プログラムを書く際に、コンパイルエラーやタイピングが軽減され利用しやすい環境である。しかし、大規模の実験での実績は少なく、実践的な学習が可能かどうかの実験も不十分である。本研究では、Fortran 言語習得を目的とする、理科系の大学生を対象に、ブロック型プログラミング学習システム「Scratckly」の実験・評価を行った。本実験では、初学者に演習型の授業において異なるインタフェースを使用してもらい、プログラミングの導入部分の評価を行った。演習の結果や採取したデータ、アンケートからブロック型のインタフェースはテキスト型インタフェースよりも初学者にとって利用した印象が良いということがわかった。

目 次

第1章　序論

本論文はビジュアルプログラミング環境を利用して、高等教育におけるプログラミング初学者が学習するためのシステムについて論じるものである。第一章では本論文のテーマを取り巻く背景、ビジュアルプログラミング環境の紹介、そして本研究の目的を説明する。

1.1　背景

近年の急速な情報化の進展により、スマートフォンやコンピュータを使用してインターネットを活用することが多くなっている。さらに AI や VR などの技術革新により、情報技術は我々の身近なものになっている [1]。

これに対応するために、実用的なプログラミングの学習は不可欠であり、大学をはじめとする教育機関でのプログラミングの教育は今や当たり前となっている。しかし、現状は高校までのプログラミング教育は学校ごとに差があり、大学入学時点でのバックグラウンドの差が他の教科に比べてかなり大きい [2]。この差は大学でのプログラミング学習の習熟度に大きく影響を与えることになる。そのため、授業に付いていけず途中でドロップアウトしたり、十分に授業内容を理解できずに修了するというケースがある [3]。

例えば、大学までほとんどパソコンを扱ったことがない人は、キーボード入力に慣れていない。このような場合、プログラムを書く際に、タイプミスがありコンパイルエラーが発生しても、どこを間違えているか分からずに修正できない。また、文字の入力に集中してしまい、授業の内容の理解がおろそかになってしまうことが考えられる。

初学者にとって、プログラミングは全くの新しいものである。４年もしくはそれ以下の期間で社会で通用する実用的なプログラミングを身につけることはかなり難しいことだと言える。

1.2 ビジュアルプログラミング環境

プログラミング教育の研究や支援システムの開発が進められる中、注目を集めているのが Squeak[4] や Scratch[5] のようなビジュアルプログラミング環境である。ビジュアルプログラミング環境はプログラミングをテキストで記述するのではなく、視覚的なオブジェクトの配置によって記述する。ビジュアルプログラミング環境はプログラムの流れが視覚的に理解しやすいことや、主としてマウスでオブジェクトの操作を行うことからタイピングのミスが生じにくく、文法エラーが発生しないという点から、特に初学者にとって利用しやすい環境である。

1.3 本研究の目的

本論文では、ビジュアルプログラミング環境を使用した初学者のためのプログラミング学習システム Scratckly の提案と評価を行う。ビジュアルプログラミング環境を使用する理由は、大学でのプログラミング学習で初学者が実用的なプログラミングを効率よく学習できる環境だからである。

今日の大学のプログラミングの講義で多く利用されているインタフェースはテキスト型のインタフェースである。一般的に企業などで利用されているものであり、大学での学習の意義を考えるとテキスト型のインタフェースを利用してプログラミングを学ぶことは納得できる。しかし、先に述べたように大学までにプログラミングの経験がない初学者にとっては初めからテキスト型のインタフェースを使用しての学習は障壁があり、効率よく学習することができない。まず、我々は初学者の障壁を明確にするために、被験者にアンケートを行い、テキスト型のインタフェースでの講義での問題点や初学者が感じているストレスについて調査した。その結果を基に、ブロック型のインタフェースで初学者が効率良く学習できるシステムを開発し、実際に講義で実験を行った。

大学のプログラミングの講義で身につけるべきスキルは、企業などで使用するテキスト表記のプログラムを理解し、実際に使用できることである。そのスキルを大学在籍期間のあいだに習得するには、ビジュアルプログラミング環境を利用し、オブジェクトでプログラミングの構造を視覚的に学びながら、テキスト表記でのプログラミング言語の文法を同時に学習する必要があると我々は考える。しかし、Scratch や Squeak のような

オブジェクトのプログラムをそのまま実行しプログラムの動きを理解するものでは、テキスト表記での言語の学習ができずに、後に、テキスト型のインタフェースへ移行する際に、テキスト表記の言語がどのオブジェクトに対応しているかを覚える過程が必要になってしまう。そこで、我々が提案するシステムはビジュアルプログラミング環境を利用して、視覚的にプログラムを理解すると同時に、テキスト表記でのプログラミング言語の文法も学習できるようにし、大学で身につけるべきスキルを学ぶ導入部分での初学者の障壁を軽減する。

第2章　関連研究

本論文ではこれ以降、Scratch[5] や Blockly[6] のように用意されたオブジェクトを組み立てることによってプログラムを作成できるビジュアルプログラミング環境で用いられる表現のことをブロック型言語と呼び、そのインタフェースをブロックインタフェースと呼ぶ。現在、ブロック型言語が注目されるようになり、ブロック型言語を用いた初学者の学習方法の研究は世界中で行われている。第 2 章では、ブロック型言語の開発と教育に関する関連研究を紹介する。

2.1　ブロック型言語の開発

本節では開発されている様々なブロック型言語を紹介する。

Brian ら [7] は Scratch を拡張する BYOB(Build Your Own Block) 機能を提案し、これまで Scratch では補えなかったプログラムの概念を学習可能にし、初学者対象の言語であった Scratch を初学者から習熟者までの学習が可能なシステムにした。しかし、大学の講義で使用するとなると、Scratch では実際のテキスト表記の言語を同時に学ぶことができないという点で利用は難しい。

石田ら [8] は PAD(Problem Analysis Diagram) と C 言語を併用した学習支援システムを開発した。このシステムは、PAD と C 言語のソースコードを相互変換できるようにし、C 言語のプログラムの文がそのまま PAD のラベルになり、PAD プログラムに変換される。

岡田ら [9] は日本語プログラミング環境「ことだま on Squeak」を用いてまずはプログラミング入門教育を行い、Java で同じ内容の授業を行うということを実践した。このシステムは日本語でプログラムが表記されているため初学者がプログラムを論理的に学ぶことができる。しかし、「ことだま on Squeak」から Java によるプログラミング学習へのシームレスな移行になっていない。

また、上記の研究 [7][8][9] では評価実験が行われていないため、学習システムとして

の有効性は確認されてない。

2.2 ブロック型言語の比較

プログラミングの学習方法として、テキストやブロックなどインタフェースの違いが与える影響を実験で比較している先行研究を紹介する。

Lewis ら [10] の研究では 6 日間のサマーキャンプに参加している学生を 2 つのグループに分け、片方は Scratch で学習し、もう一方は Logo を使用して学習した。この実験では、音楽、映画、ゲームを作ることを学習目標にしている。結果は仮説に反してどちらのグループも同じように演習が困難だった。Logo を使用した学生は実験後にコンピューティング能力に自信を持った。また、それぞれ特定のプログラムの記述において成果を出し、Logo の学生はループ、Scratch の学生は条件をよく理解していた。

松澤ら [11] はブロック型言語とテキスト型言語を併用してプログラムを書くことができる「BlockEditor」を開発した。このシステムは、ユーザ自らがブロックインタフェースを使用するかテキストインタフェースを使用するか選択でき、その両方を使用してプログラムを書くこともできる。実験は 110 名のプログラミング初学者に 14 週間の講義の中で使用してもらい、評価を行った。初学者は高い割合でブロック型言語を使用し、講義が進むにつれてブロック型からテキスト型に移行していくことがわかった。しかし、この研究ではブロック型言語の使用頻度や使用量、アンケートを基にしたブロック型言語の感想の評価が中心であり、正解率や習熟度に関しての調査はされていない。

Booth ら [12] は成人を 2 グループに分け、テキスト型言語 (Java) とブロック型言語 (Modkit) で学習した。この研究では新たなプログラムを作成するのではなく、既存のプログラムを修正する活動で良い結果が出た。Modkit グループの方がユーザフレンドリーであり、作業負荷が低く、より高い正解率であった。しかし、サンプルサイズが小さく、統計分析も行っていない。

Dann ら [13] は大学の学部生の入門コースの生徒を対象に、Alice3 を使用し学習した後に Java に移行する講義を行い、コース終了時にテストを実施し、過去のテキストで単純に Java を学んだクラスと比較した。Alice を利用した年は過去に比べ、テストの得点が高くなった。

2.3 参考研究

我々の研究をするにあたって参考になるデータ、実験を行っている先行研究を紹介する。

McKay ら [14] は、Scratch、Alice、Green-foot、LEGO Mindstorms NXT、Python など、教育に使用されるさまざまなブロックおよびテキストプログラミング環境を比較するための予測モデリングシステムで実験を行った。CogTool というプロトタイピングツールを使用して、各環境でさまざまなプログラミングタスクの実行時間をモデル化した。結果は、テキスト言語が挿入や置換などに適していることを示し、ブロック言語は削除や移動に適していた。しかし、モデルがプログラムを書く際の思考に費やされた時間を考慮しておらず、さらに差がひらく可能性がある。

Thomas ら [15] は、過去のビジュアルプログラミング言語の研究において、どの要素がその成功に起因しているかを明確にするために、その一番の特徴であるブロックでの入力のみを取り出し、テキストの入力での学習と比較した。入力方法以外は全て同じになるようなインタフェースを作成し、初学者をテキストインタフェースを使用するグループとブロックインタフェースを使用するグループの２つのグループに分けて実験を行った。結果はブロックインタフェースのグループの方が課題の達成、効率においてパフォーマンスが向上した。

2.4 関連研究まとめ

この章では既存のブロック型言語のシステムや実験を紹介した。このことから分かるようにブロック型言語は初学者にとって有効なプログラミング学習手段である。学習方法には大きく２通りあり、(1) ブロックインタフェースで学習し、テキストインタフェースに移行する方法 (2) ブロックインタフェースのみで学習を進める方法がある。既存のシステムではブロックでプログラムを理論的に学ぶと同時にテキスト表記によるプログラミング言語の文法を覚えるということが難しい。また、そのようなシステムの有効性を評価できていない。我々は初学者がより効率的に学習するために、ブロック型言語でプログラムを作成しながら、テキスト表記のプログラミング言語を学習できるシステムの提案および評価を行う。

第3章　システムの設計

第 3 章では「Scratckly」の機能を説明する。

3.1　「Scratckly」と機能の概要

本研究では初学者を対象とした演習形式でのブロック型プログラミング学習システム「Scratckly」を提案する。Scratckly は「Google Blockly」を基に開発し、初学者のプログラミング入門でのプログラミングの概念や構造を視覚的に学ぶと同時にテキスト表記のプログラミングの文法も学ぶことができ、効率的な学習を支援するシステムである。Scratckly の対象者として想定している学習者は高等教育機関の学生のプログラミング経験が無い、もしくは経験が浅い（初学者）である。本システムは大きく以下の 3 つの機能を持つ。

機能 1:ブロック型言語でのプログラム記述機能（ユーザはブロック型言語を用いてプログラムの記述が可能）

機能 2:ブロック型言語の Fortran への変換（ブロック型言語で記述したプログラミングがリアルタイムで Fortran に書き換えられ、同一画面で確認可能）

機能 3:演習型授業での効率化（ユーザは課題の確認、プログラム作成・実行、プログラムの正誤判定が可能）

機能 1 は、キーボードを使用してテキストを入力する際、初学者は、タイピングの遅れやコンパイルエラーが起こることで、困惑してしまい苦手意識を持ってしまうということを防止する。これは一般的なブロック型言語の利点であるが、我々の Scratckly においてもその機能が失われないことを考慮している。

機能 2 は、ユーザがプログラムの概念と基本構造を視覚的に学ぶと同時に、テキスト型言語を学びテキストインタフェースでの学習で得られる知識も習得することを考慮している。

機能 3 は、限られた時間の中で効率よく学習するという目的に基づいて、プログラムの作成以外の部分を容易に行えるようにした。

3.2　機能 1:ブロック型言語でのプログラム記述機能

3.2.1　インタフェース

開発したシステム「Scratckly」の外観を図 3.1 に示す。図 3.1 上部の黄色の領域には解くべき課題が表示されている。その下部には 2 つのボタンがある。このボタンについては 3.4 で説明する。さらに下部にあるのが作業ウィンドウで、左のウィンドウにブロック型言語でプログラムを作成すると、右のウィンドウに Fortran に変換されたプログラムが表示される。Fortran のウィンドウの下部にはプログラム実行のための入力部と実行ボタンがある。

作業ウィンドウの左側にある各カテゴリから必要なブロックを取り出し、作業ウィンドウでブロックを組み立てる。

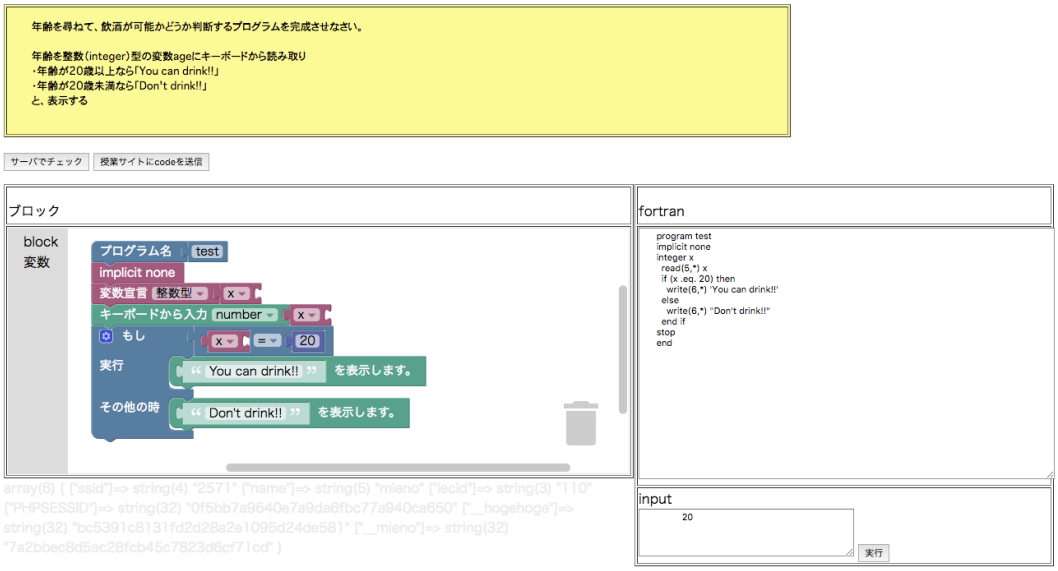

図 3.1: Scratckly のインタフェース

3.2.2 ブロックの設計

Scratckly の基礎部分は Google Blockly[6] を利用して開発した。ブロック設計の際に考慮した点は以下の２つである。ひとつめはブロックラベルの日本語化である。初学者がつまずく要因として、プログラム言語が母語と異なることにより、難しく感じることや命令の内容が理解しにくいことが挙げられる。日本人がよりプログラムを理解しやすいようにブロックラベルを日本語に変更した。

ふたつめは Fortran への変換を可能にしたことである。従来の Blockly には Fortran への変換機能は準備されておらず、変換を可能にするには設定の変更や新たなブロックの作成が必要であった。作成したブロックを図 3.2 に示す。Google Blockly には変数宣言のための機能が準備されていない。Fortran は変数を宣言せずにプログラムを作成することが可能であるが対象者が初学者であることを踏まえると、変数宣言の概念を理解することは重要であると考えられる。そこで我々は、変数宣言を可能にするために「変数宣言ブロック」を作成した。変数の型は現在、整数型・単精度実数型・倍精度実数型・複素数型が利用できる。変数の型は「変数宣言ブロック」のドロップダウンボタンをクリックすることで容易に変更できる。「implicit none」や mod 関数による剰余の計算など、Fortran での特徴的な機能も新たに追加する必要があった。また、Blockly に同じような意味を持つブロックが存在したが、テキスト表記に変換する際に、順序が逆になったり、ブロックに必要のない連結や他のブロックを入れるための穴があるなどユーザの混乱を起こす可能性のあったブロックについても Fortran の表記に一致するように変更した。

3.3 機能 2:ブロック型言語の Fortran への変換

3.3.1 ブロックから Fortran への変換

Scratckly はブロック型言語で組み立てられたプログラムを Fortran のソースコードに変換する。この変換システムによってユーザは自分が組み立てたブロック型のプログラムが Fortran ではどのように記述されるかを確認できる。

Blockly は Javascript や Python などへの変換機能を備えている。今回の実験で使用する Fortran は Python としてテキスト表記に変換されていたプログラムを変換過程で Fortran のテキスト表記に変換するように変更して表示した。実際にブロックから変換されて表示

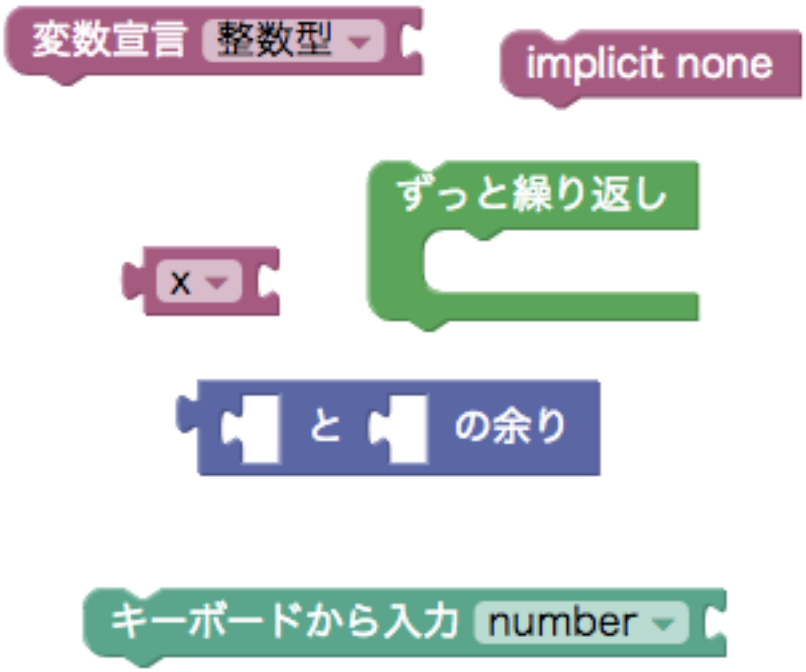

図 3.2: 新たに追加したブロック

されたプログラム（年齢を入力して、飲酒可能かどうかを判定）を図 3.3 に示す。Fortran の文法に沿うように各行の先頭に 6 個のスペースを挿入しており、ソースコードの可読性を高めるためのインデントも自動的に行う。

3.3.2　リアルタイム変換

ユーザがブロックの追加や削除、位置や構造の変更を行うと、Scratckly は対応する Fortran のソースコードをリアルタイムに更新する。そのため、ユーザはブロックで構造を視覚的に確認し、その動作がどのようにテキストで表現されるのかをソースコードのウィンドウを見るだけで確認できる。リアルタイムで表記することでブロックの画面、テキストの画面を同時に確認し、学習効率を高めることを考えて設計した。

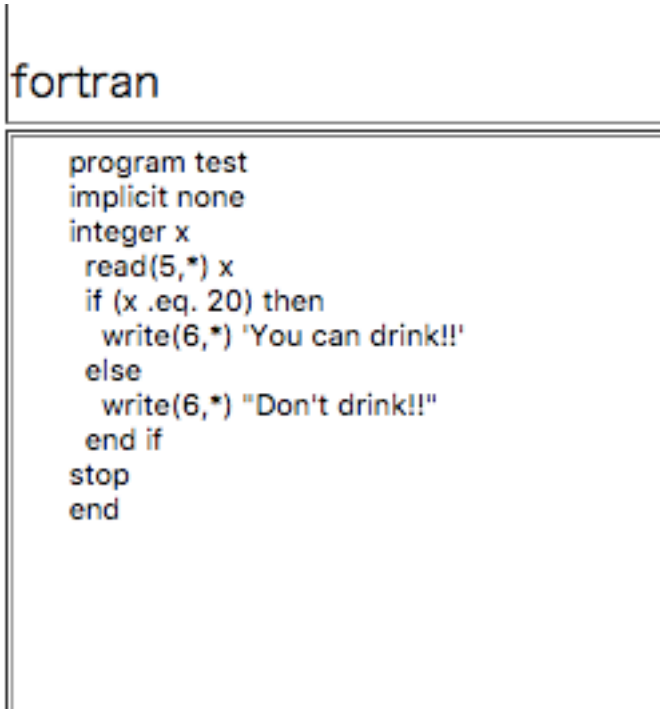

図 3.3: 変換された Fortran のソースコード例

3.4 機能 3:演習型授業での効率化

演習型の授業では考える時間を設けるためにプログラムの思考・作成以外の効率化が重要である。Scratckly は同一画面で問題の表示、ブロック型プログラムの作成、テキスト型プログラムの表示・実行、実行の際に必要な入力、プログラムの正誤判定が可能である。

プログラムの実行例を図 3.4 に示す。図 3.4 に表示されているプログラムは図 3.3 で示したプログラムである。Fortran のソースコードが表示されている下部に「input」という表記、テキストボックス、「実行」というボタンがあるウィンドウがある。このウィンドウのテキストボックスの中に、変数に入力したい値を入力し、実行ボタンをクリックすると図 3.4 のように画面上部に入力された値と出力が表示される。

ユーザは何度か実行を試し、出力が正しければ、問題が表示されたウィンドウの下部にあるボタンでプログラムの正誤判定ができる。図 3.5 に示すように、「サーバでチェック」と表記されたボタンを押すと、プログラムが実行される。その出力によって正しい

プログラムであるかが判定され、画面上部に「正しい出力です。」と表記される。空白の有無など、ユーザによってプログラムの出力が多少異なることを考慮して出力の判定を行い、誤判定を防止している。

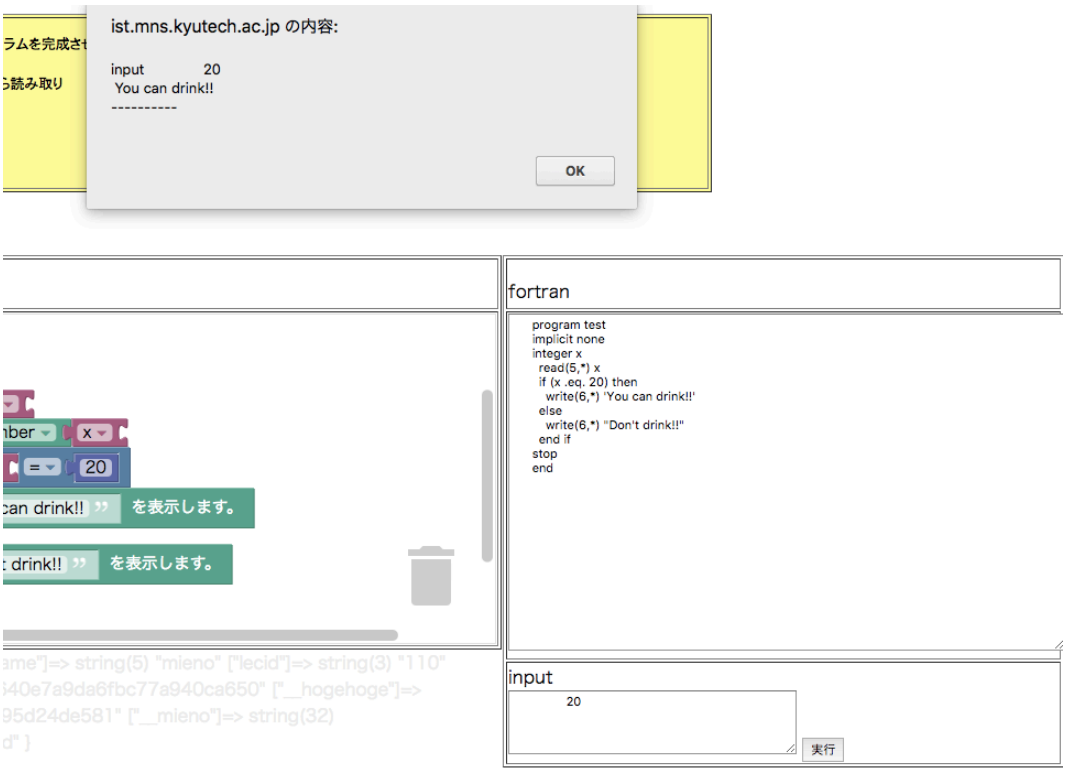

図 3.4: プログラム実行の時の画面

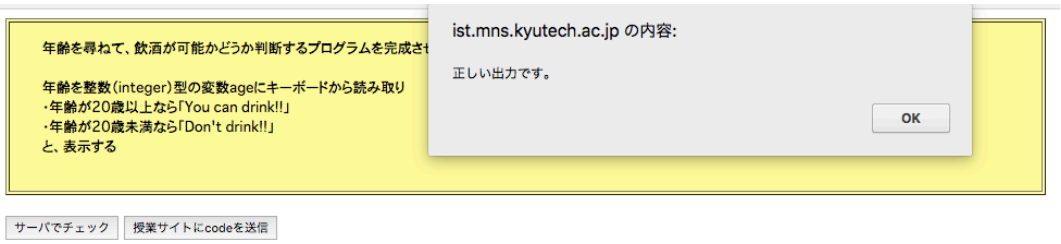

図 3.5: 正誤判定時の画面

第4章　実験

第 4 章では実験環境、実験方法を説明する。

4.1　実施環境

本論文の実験は著者所属大学・学部で 2017 年度後期の授業で実施された。実施環境の概要を表 4.1 に示す。プログラミング初学者 218 名を対象に条件分岐と繰り返しに関する問題を 90 分間の演習形態の授業で行った。ティーチングアシスタントは教員 1 名と学生 2 名の計 3 名が配置され、実験時間の 90 分のあいだ学生は自由に質問ができた。学生が使用するインタフェースの割り当てはクラスごとに行い、インタフェースの違いがクラス内で起こり学生が混乱するのを避けた。また、実験に参加した学生はテキストインタフェースでの 90 分 $\times$ 10 コマの演習形式での Fortran の授業を受講済みであり、テキストインタフェースに慣れている。

4.2　実験方法

4.2.1　事前アンケート

我々はブロック型言語でのプログラミング学習の実験を行う前に、学生に対して事前のアンケート調査を行った。アンケートの内容は以下の 7 問である。

- プログラミングは得意であるか
- プログラミングの経験はあるか
- 大学でのプログラミングの授業は理解できているか
- タイピングや指示された作業についていけないことがあるか
- コンパイルエラーを修正できないことはあるか
- 理解が不足していると思う内容はどこか（自由記述）

表 4.1: 実験の実施環境

対象学部・学年	工学部（理科系）・2 年
対象授業	情報処理応用（プログラミング）
クラス数	3 クラス（A:80 名 B:59 名 c:79 名）
前提条件	テキストでの Fortran の学習を 90 分 × 10 コマ受講済み
実験時間・形式・内容	90 分・演習形式・条件分岐、繰り返し
指導体制	教員 1 名、アシスタント 2 名

● これまで困ったことがあるか（自由記述）

この事前アンケートはテキストインタフェースを使用した現在の学習での学生の習熟度やプログラミングに対する意識を調査するためのものであり、235 件の有効な回答を得た。実験の対象人数と異なるのは事前アンケートは実験日の 1ヶ月前に行ったためであり、実験のデータは事前アンケートと実験授業の両方に参加した学生のみを対象にしているためである。

4.2.2 実験

実験は、各グループごとに 90 分ずつ行った。実験のタイムテーブルを表 4.2 に示す。

まず、授業開始から 20 分間は、実験の進行の流れや回答方法、インタフェースの操作方法などの説明を行った。授業開始から 20 分が経過すると「条件分岐」の内容の演習問題が回答可能になり、学生は自分のペースで解くことができる。その後、授業開始から 40 分が経過した時点で、「条件分岐」の演習問題は回答不可になり、「繰り返し」の内容の演習問題が回答可能になる。授業開始から 60 分が経過すると「繰り返し」の演習問題は回答不可になり、最後のテストが回答可能になる。テストは授業開始から 80 分の時点で回答不可になり、ブロックインタフェースを使用した、A、B のグループはテストが終了した学生からアンケートに回答する。

演習問題は各 3 問用意されており、1 問目は最も難易度が低く、3 問目が最も難易度が高いというように順に難易度が上がる。テストは 2 問用意されており、1 問目よりも 2 問目の難易度を高く設定した。

実験後のアンケートの内容は以下の 5 問である。

ブロック型言語での学習について

● プログラミングの苦手意識がなくなると思うか

表 4.2: 実験のタイムテーブル

時間（分）	内容	問題数
0 〜 20	実験の説明	2
20 〜 40	演習 1（条件分岐）	3
40 〜 60	演習 2（繰り返し）	3
60 〜 80	テスト（条件分岐、繰り返し混合問題）	2
80 〜 90	アンケート	

● 授業に遅れることがなくなるか

● 授業内容の理解が深まるか

●これまでの授業に比べて良くなると考えられる項目を選択してください（複数回答可）

・コンパイルエラーが起こりにくい

・授業に置いていかれることがなくなりそう

・内容を理解しやすい

・タイピングが少なくてやり易い

● 自由に感想を書いてください

このアンケートはブロックプログラミングに対しての意識調査として実施した。A グループ 79 名、B グループ 59 名の有効回答を得た。

4.3 使用するデータ

4.3.1 実験での問題とデータ

本実験では 2 つの問題が発生した。1 つ目は、A グループの実験を行う際にサーバの負荷が高くなり、3.4 で説明した、正誤判定の際に、画面が固まりプログラミングに費やす時間が削られた。これは、プログラムを送信する際に、負荷を軽減して処理を行っておらず、一度に大勢がアクセスしたためである。この問題を考慮し、問題の影響を受けたと見られる学生のデータを問題ごとに選定し、実験のデータから除いた。

２つ目は、実験の繰り返しの内容の問題 3 問中 1 問の答えが誤っており正誤判定の際に、正解にも関わらず不正解となってしまう学生がでた。このことを考慮し、正誤判定に問題があった設問をデータから除いた。

4.3.2 ログの記録

本実験では学生がアクションを起こすたびに、タイムスタンプが押され、そのデータがデータベースに記録される「ログ」によって学生の進度に関するデータを採取した。記録されるアクションはインタフェースによって異なり、ブロックインタフェースでは、3章で説明した画面上の全てのウィンドウ、ボタンに何らかのアクションを起こすとタイムスタンプがデータベースに記録される。テキストインタフェースでは、キーボード上の特定のキーを押すことでタイムスタンプがデータベースに記録される。インタフェースによってデータの取得方法が異なることを考慮し、ログによって取得したデータは学生の作業時間の分布のデータとして利用する。Q2, Q3, Q4 においては 4.3.1 節でも述べた理由により、ブロックインタフェースのログが正しく取得できていなかった。また、ログインタラクションで取得したデータを元に、個人的に何らかの問題が発生したといえる学生をデータから除いた。

上記の理由によって実験のデータから除かれた学生は表 4.1 の実験対象人数に含まれていない。

第5章　結果

第 5 章では 4 章で述べた実験の結果を示し、考察を行う。

5.1　事前アンケート

事前アンケートは実験前までの学習での習熟度やプログラミングに対する意識を調査するために行った。結果としては、現状の学習方法ではほとんどの学生が何らかの不満を抱えるという状況だと判明した。アンケートの結果をグループごとにまとめたものを図 5.1 に、自由記述欄に記述されたものを表 5.1 に示す。図 5.1 を見ると、現在のプログラムング学習方法では、プログラミングに苦手意識を持っている学生はどのグループにおいても 90 %近くの割合を占めているということがわかる。やはり、先に述べたようにテキストインタフェースでのプログラミング学習は初学者にとって、障壁が多いようである。また、どのグループにおいても、自分自身のタイピングの速さや正確さ、コンパイルエラーの回避や修正に対して否定的な意識を持っている学生が 70 %を占めている。このような原因から苦手意識が起こっていると考えられる。プログラミングの授業で困ったことを自由に記述してもらった結果では、やはり多かった答えとして、「コンパイルエラーが修正できなくて、授業がそのまま進んでしまう」や「タイピングが遅くて説明についていけない」が挙がる。実際の授業をみてもそのような学生がいるように感じる。

このような結果から、現状行われている学習方法では不十分であるということが言える。

5.2　実験の結果と評価

まず、実験の各問題に対する回答数を表 5.2、実験中のログの時間分布を図 5.2、各グループの各問題の正解率の比較を図 5.3 に示す。また、提出された各回答について、4 点満点で採点を行った。採点方法は以下の通りである。

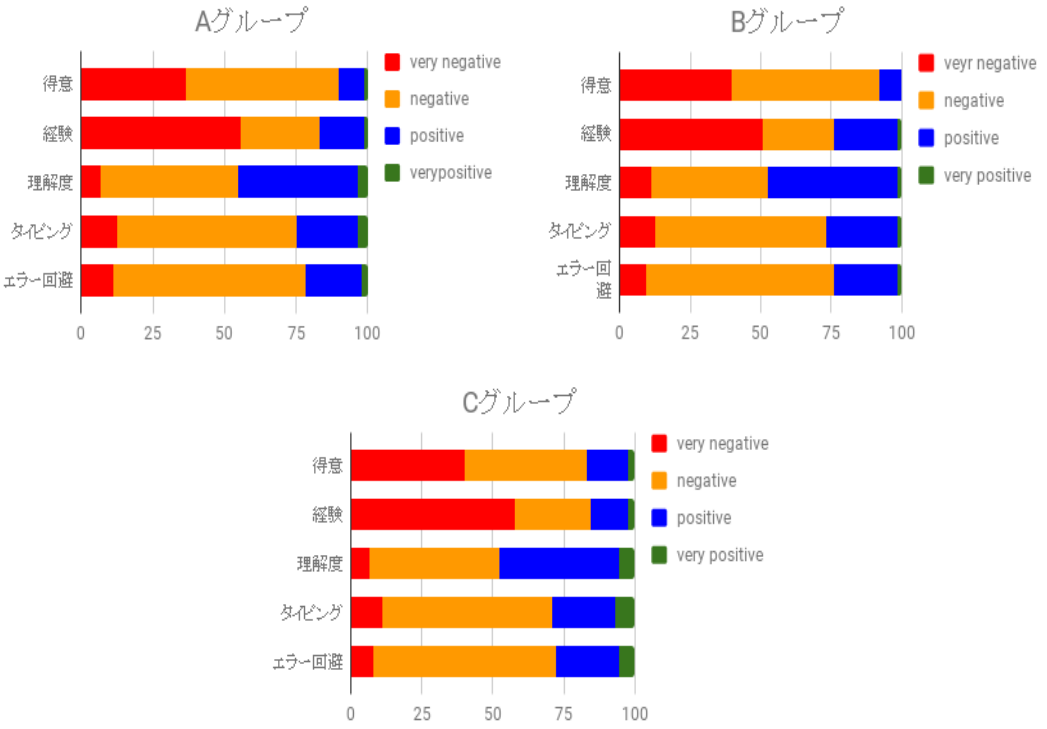

図 5.1: 事前アンケートの結果

● 4 点：正解している

● 3 点：実行エラーで不正解

● 2 点：コンパイルエラーで不正解

● 1 点：回答途中で時間切れ

この採点を行ったのは、回答の時間、コンパイルができているか、実行結果は正しいかという順に分けて点数を付けることで、インタフェースの有効性が明らかになると考えたからである。インタフェースごとの各問題の評点の平均を図 5.4 に示す。

5.2.1 演習 1（Q2、Q3、Q4）の結果と評価

演習 1 の問題は Q2, Q3, Q4 の 3 問で、すべて条件分岐に関する問題である。

まず、回答数をインタフェースごとに比較すると、表 5.2 からテキストインタフェースでは Q2 の回答数が 72 名で Q4 が 16 名となっており、Q4 を回答した学生が約 20 % に

表 5.1: 自由記述欄のコメント

回答者	コメント
A	コンパイルエラーを修正できない時が一番大変な気がする。
B	タイピングが遅いのでついていくのは大変。
C	エラーが表示されていても、何が間違っているか分からない。
D	一度つまるとどんどんわからなくなる
E	自分のタイピングが遅くて、分からなくなる

とどまっているが、ブロックインタフェースでは Q2 の回答数が 117 名で Q4 が 60 名と約 50 %の学生が Q4 を回答していることがわかる。このことから、条件分岐を学ぶ際に、ブロックインタフェースを利用することによって、タイピングによる負担の軽減などが起因して、初学者がスムーズに回答することができるといえる。また、テキストインタフェースを使用した学生の Q4 の回答数が著しく減少している理由として、事前アンケートでもあったように、エラーなどのつまづきで、諦めてしまっているのではないかと考えられる。しかし、どちらの学生も半数近くが Q4 に達していないということは、実験自体に何らかの問題があったということも考えられる。

図 5.3 から、Q2, Q3 においてブロックインタフェースを使用した学生の正解率が 76.9 %, 58.3 %であるのに対し、テキストインタフェースを使用した学生の正解率は 83.8 %, 80.4 %となり、ブロックインタフェースを使用した学生の方が正解率が低いことが分かる。これは、テキストインタフェースを使用している学生は実験以前の授業でインタフェースに対する習熟度があり、初めから問題に対して積極的に取り組めたが、ブロックインタフェースを使用した学生は、初めて扱うインタフェースであり、インタフェースに十分慣れていないため操作が困難であったのではないかと考えられる。また、実験の Q2, Q3 はたくさんの if 文を繋ぎ条件を設定する必要がある問題であったため操作を誤り、思わぬエラーが発生し、正解率が低くなってしまったと考えられる。しかし、Q4 の正答率はブロックが 36.7 %、テキストが 18.75 %と逆転し、ブロックインタフェースを使用した学生の方が正答率が高くなっている。これは、ある程度ブロックインタフェースに慣れてきたことによって、問題に集中することができるようになったためと考えられる。さらに、正解率がテキストインタフェースの学生よりも高くなっていることは、ブロックインタフェースの特徴である視覚的に構造が理解しやすい点や入力が容易という点が起因し、その特徴を活かしながら問題に回答したためだと考えられる。

評点については図 5.4 から、正答率と同じように Q2, Q3 ではテキストインタフェースの方が高いが、Q4 では逆転している。

表 5.3 より、演習 1 の評点では、Q2, Q4 でブロックインタフェースとテキストインタフェースの有意差は出なかったが、Q3 においては有意水準 5 %において Welch の t 検定を行ったところ、評点の平均値において、ブロックインタフェースよりもテキストインタフェースのほうが有意に高いという結果となった (t(96.9) = 2.25, p = 0.027)。

5.2.2 演習 2（Q6、Q7、Q8）の結果と評価

演習 2 の問題は Q6, Q7, Q8 の 3 問で、すべて繰り返しに関する問題である。Q7 においては先に記述した通り、正誤判定に不備があったため評価から除外した。

まず、回答数について表 5.2 より、どちらのインタフェースを使用した学生も Q8 を回答した学生は Q6 の半数以下となっている。これは、問題 7 でつまづいたという可能性が考えられるため Q6 についてログの変化と正解率を評価する。

図 5.2 の Q6 のログの変化に注目すると、テキストインタフェースでは 5〜9 分のログの回数と 10〜14, 15〜19 分までのログの回数がほとんど変わっていない。これは Q6 を 15〜19 分の時間まで解き続けているということであり、演習 2 の多くの時間を Q6 に費やしているということがいえる。ブロックインタフェースの Q6 のログの回数は 5〜9 分より 10〜14 分の時点で半数ほどになってその後も減少していることから、ブロックインタフェースを使用した学生は Q6 を 10〜14 分の時点で解き終わり始めているということがわかる。繰り返しに関する Q6 においてはブロックインタフェースの方が、テキストインタフェースよりも効率よく回答できたといえる。

図 5.3 より、Q6 はブロックインタフェースの正解率が 92.5 %、テキストインタフェースの正解率は 86.7 %、Q8 はブロックインタフェースの正解率が 9.5 %、テキストインタフェースの正解率が 9.5 %となっており、ブロックインタフェースを使用した学生の方が正解率においても高い割合を占めている。インタフェースに対する習熟度の面でブロックインタフェースを使用する学生の方が劣っているということを踏まえると、繰り返しに関する問題を解くにあたって、ブロックインタフェースを利用する方が効率的に正確にできるということがいえるのはないかと考えられる。また、繰り返しのプログラムを書く際に Fortran では終わりに「end do」と記述する点や、条件の書き方が独特な点など

少し特徴的な表現があることから、ブロックインタフェースでは特徴的な部分を自動で記述するのでミスを避けられたとも推測できる。

図 5.4 においても同じような結果がみられ、Q6 の評点の平均はブロックが 3.85、テキストが 3.73 となっている。また、表 5.3 より演習 2 においても Q6, Q8 の評点の平均値において、インタフェースの違いによる有意な差は出なかった。

5.2.3 テスト（Q9、Q10）の結果と評価

テストの問題は Q9, Q10 の 2 問で、条件分岐と繰り返しの混合問題で、演習に比べてやや難しい問題である。表 5.2 より Q10 における正解者が 2 名という点から時間設定、図 5.2 のログの回数を見てもほとんどログがないことから回答が困難であったと考えられ、問題の難易度にも問題があったと推測される。このため、Q10 の評価は難しい。よって、Q9 について評価を行う。

図 5.3 より Q9 のブロックインタフェースの正解率は 66.7 %で、テキストインタフェースは 55.2 %となっておりブロックインタフェースの方が正答率が高い。複雑なプログラムになったことで視覚的に構造が確認できるというブロックの利点がこの結果に関係しているのではないかと考えられる。図 5.2 の Q9 に注目すると、ブロックインタフェースの学生はログの数が 10〜14 分までには減少が見られず、15〜19 分のところで半分に減っていることから 10〜19 分の間に多くの生徒が解き終わっていると推測される。テキストインタフェースの学生は 10〜14 分のログが大きく減少していることからこの範囲で多くの生徒が解き終わっていると推測できる。しかし、正解率がブロックインタフェースを使用した生徒の方が高いという点となっていることから、ブロックインタフェースの方が正確にプログラムが書けているといえる。

図 5.4 から評点の平均は Q9 において、ブロックインタフェースが 3.56、テキストインタフェースが 3.43 となっている。Q10 においてブロックインタフェースは 1.29、テキストインタフェースは 2.07 とテキストインタフェースの方が高くなっていることからテキストインタフェースは Q10 が正解にはならなかったものの取り組むことができていることがわかる。

表 5.3 よりテストにおいても、Q9 の評点に対して有意差は出なかったが、Q10 においては、有意水準 5 %において Welch の t 検定を行ったところ、評点の平均値において、ブ

表 5.2: 問題と回答数と正解数

問題	Q2	Q3	Q4	Q6	Q8	Q9	Q10
回答数（ブロック）	117	84	60	134	65	72	49
正解数（ブロック）	90	49	22	124	18	48	0
回答数（テキスト）	72	46	16	75	21	76	15
正解数（テキスト）	60	37	3	65	2	42	2

ロックインタフェースよりもテキストインタフェースの方が有意に高いという結果となった (t(18.7) = 2.54, p = 0.020)。

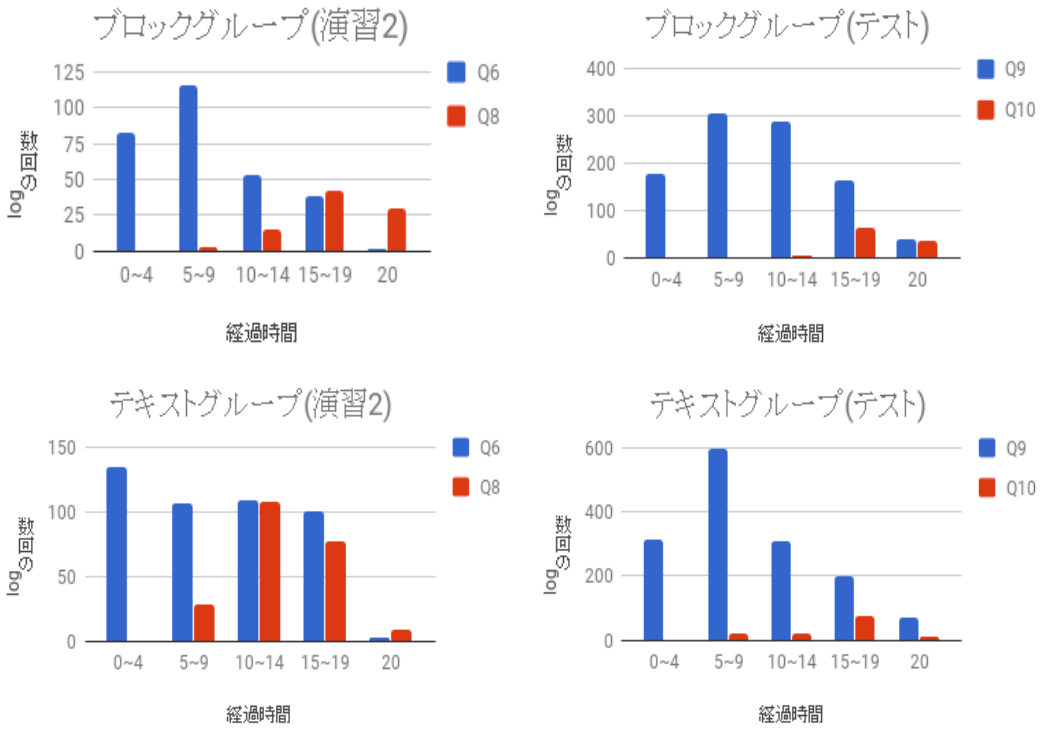

図 5.2: ログの時間分布

5.2.4 実験後アンケート

実験の最後に行った実験アンケートではブロックインタフェースで実際に学習した印象を調査したものであり、ブロックインタフェースを使用した B, C グループにのみ実施

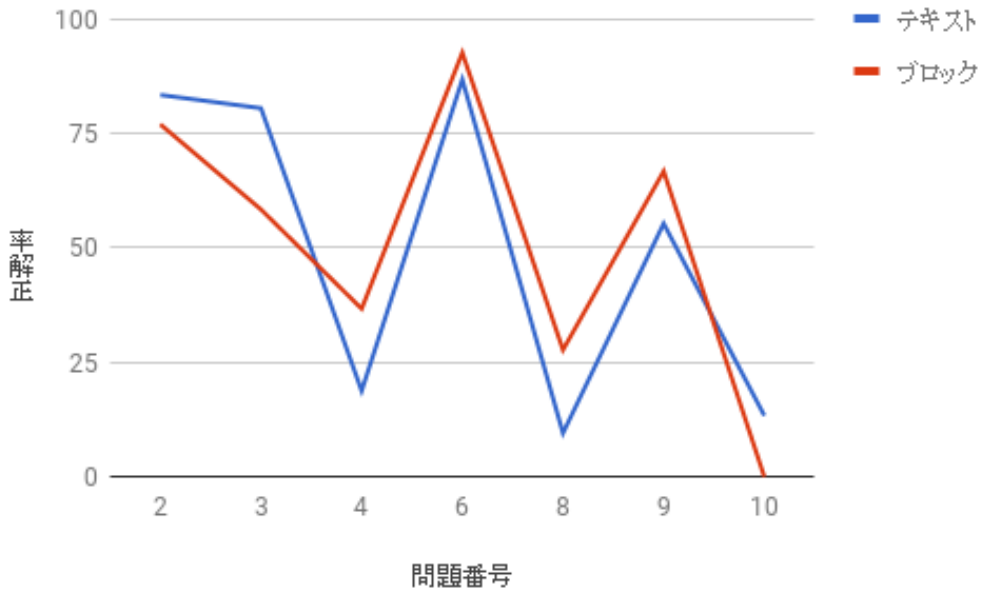

図 5.3: インタフェースによる各問題の正解率の比較

表 5.3: 評点の検定結果

問題	Q2	Q3	Q4	Q6	Q8	Q9	Q10
t 値	0.51	2.25	1.18	0.97	1.25	0.85	2.54
確率 p	0.609	0.027	0.251	0.335	0.219	0.399	0.020

された。アンケートからブロックインタフェースでの学習に対する利点の感じ方が学生間で別れるという結果が得られた。アンケートの結果を表 5.4 に示す。演習やテストの結果だけを見るとブロックインタフェースとテキストインタフェースに差がないように感じられるが、アンケートの結果では興味深いことがわかった。アンケートの「これまでの授業に比べて良くなると考えられる項目を選択してください」という複数選択可能の項目で、1 つ以上選択し、改善点があると回答した学生が 94 %いた。また、学生がアンケートで改善点として選択した項目は学生ごとにばらつきがあった。テキストインタフェースに比べて、苦手意識が改善されたり、内容が理解しやすくなると回答した学生は 65 %程度であり、コンパイルエラーが起こりにくいことやタイピングが少ないことに起因して、授業に遅れをとることが少なくなると回答した学生は 91 %いた。アンケートの結果から、ブロックインタフェースに対して初学者は良い印象を持っていることがわ

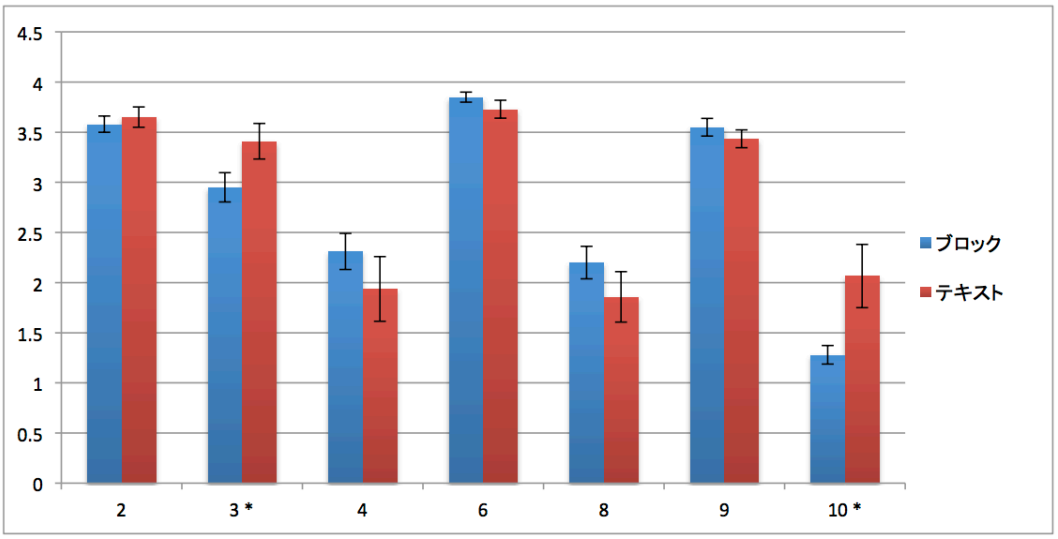

図 5.4: 評点の平均点と標準誤差

かる。しかし、演習の結果が良くなかったということは実験自体の問題が原因であると考えられる。

表 5.4: 実験後アンケート結果（有効回答数：138）

回答内容	人数	割合
苦手意識が改善される	95	68.84 %
遅れることがなくなる	125	90.58 %
内容を理解しやすい	92	66.67 %
テキストインタフェースより 1 つ以上の改善点あり	130	94.20 %

5.3 全体的な結果と評価

先に述べたように演習 1・2、テストにおいてブロックインタフェースの使用による問題の回答効率や回答の結果に影響がなかった。。しかし、各演習、テストの正解数を見てみると、テキストインタフェースは Q4, Q8, Q10 について 3 名, 2 名, 2 名となっておりいずれも同じ人物が正解している。しかし、ブロックインタフェースの方は各演習、テストのこの 3 問において多くの学生が正解している。この 3 問は他の問題に比べ少し難易度が高くなっている。このことから、テキストインタフェースを利用すると、一部の理

表 5.5: 自由記述欄のコメント（実験後アンケート）

回答者	コメント
A	ブロックによってプログラミングが解釈しやすくなった。
B	ブロックでプログラミングをすることで条件などをより明確にして実行に移すことが可能だと感じた。
C	タイピングが少なくてやりやすかったです。
D	いつもより少し楽にできた。

解している学生は問題に対応して回答することができる。しかし、多くの学生は、少し難しい問題になると解けなくなる。ブロックインタフェースを利用すると、難しい問題でも考えやすくなるということがいえる。正解率や評点からも同じような結果が見られる。また、実験後に行ったアンケートの自由記述欄のコメントの肯定的な意見を表 5.5 に示す。アンケートで確認した学生のブロックインタフェースに対する良い印象が実験の結果に反映されてない理由として、実験自体の問題があったと考えられる。

第6章　まとめ

本研究では、高等教育におけるプログラミング教育の現時点での問題を解決するために、プログラミング初学者のためのプログラミング学習システムの提案と実践を行った。事前に行ったアンケートからはプログラミング学習の現状は良いとは言えない結果となった。大学に入学してくる学生のほとんどが、プログラミングの経験がないまたは少ないということが原因で、プログラミングの授業についていけずに苦手意識があるということがわかった。また授業についていけない大きな要因として、タイピングの遅れやコンパイルエラーの修正ができないことが挙げられていた。そこで我々はビジュアルプログラミング環境を使用することでその問題を解決しようと考え、ブロックインタフェースを使用した「Scratckly」を提案し、実験を行った。

今回の実験により、ブロックインタフェースを使用することで、初学者はプログラミングに対して、テキストインタフェースを使用するよりも好印象を持ってプログラミングに臨むことができるということがわかった。

しかし、実験の結果から、このシステムは現在のプログラミング学習における初学者の抱える問題を解決できるとは言い難い。このような結果になった原因は実験後のアンケートから推測すると実験方法自体に多くの問題があったと考えられる。

まず、時間設定や実験の形式である。実験を行うにあたって、ブロックインタフェースとテキストインタフェースの間に実験結果の差を出すために、時間設定を少し短めに設定し、問題を回答できる学生とできない学生がでるようにした。しかし、初学者はインタフェースに関係なく問題を素早く解くということは困難であった。さらに、ブロックインタフェースを使用する学生は、その日初めて扱うものであったため、早く解くということができず、インタフェースに対する習熟度の差が結果に大きく影響していると考えられる。アンケートの自由記述欄にも次のような意見が挙げられていた。「理解はしやすかったが、問題数が多かったので、時間がもう少しあればいいと思った。」「制限時間少なすぎます、どれもできたのに提出までいけなかった。」「ブロック自体は簡単でし

たが、進むのがはやかったです。」このような意見から時間の設定によってはより良い結果が得られたかもしれない。

また、ブロックインタフェースを使用した学生は実験以前にテキストインタフェースを使用して多くの時間学習を経験しており、その学習に必死で付いていっているという状況で、いきなりブロック型言語での演習をさせられたことで、抵抗感が発生したということも考えられる。アンケートに自由記述欄にも「普通にプログラミングするほうが慣れているのでやりやすかった」や「慣れていないので、逆に授業に遅れをとってしまった感覚だった。」という意見があり、学生の変化に対する意識が影響を与えているということも考えられる。

このような問題点から本来のインタフェースがもつ特性を十分に検証できたとは言い難い。システムを改善し、よりインタフェースの特性が活きるような実験の方法で検証を行う必要がある。引き続き、ブロックインタフェースの有効性を検証していきたい。

謝辞

卒業論文を完成するにあたり、ご指導ご教授くださりました三浦准教授に御礼申し上げます。また、輪講や中間発表においてご指導やご教授を下さりました情報セクションの先生方に御礼申しあげます。加えて、本論文の評価実験において、被験者としてご参加頂きました九州工業大学の学生、アシスタントとしてご参加頂きました情報セクションの学生にお礼を述べたいと思います。最後に、私の意思を尊重してくださり大学進学を応援して頂き、経済面や生活面において、ご支援をして頂いた家族に心から感謝申し上げます。

参考文献

[1] 文部科学省. 「2020 年代に向けた教育の情報化に関する懇談会」中間取りまとめ. http://www.mext.go.jp/a_menu/shotou/zyouhou/1369536.html, 2016.

[2] 文部科学省. 情報教育の現状と課題、改善の方向性（検討素案）. http://www.mext.go.jp/b_menu/shingi/chukyo/chukyo3/004/siryo/07092002/007.html, 2016.

[3] 岡本雅子. 模倣の重要性に着目した初学者向けプログラミング教育の研究. 2014.

[4] OSDN 株式会社. squeak. https://ja.osdn.net/projects/squeak-ja/.

[5] MIT. scratch. https://scratch.mit.edu.

[6] Google blockly. https://developers.google.com/blockly/.

[7] Brian Harvey and Jens Mnig. Bringing“no ceiling”to scratch: Can one language serve kids and computer scientists? In *Constructionism 2010*, 2010.

[8] 石田真樹, 桑田正行. C プログラミングの学習支援に関する研究 － pad エディタを用いたアルゴリズム学習支援システムの構築－. 情報処理学会研究報告コンピュータと教育（CE）, pp. 41–48, 2000.

[9] 岡田健, 杉浦学, 松澤芳昭, 大岩元. 日本語プログラミングを用いた論理思考とプログラミングの教育. 情報処理学会研究報告コンピュータと教育（CE）, pp. 123–128, 2007.

[10] Colleen M. Lewis. How programming environment shapes perception, learning and goals: logo vs. scratch. In *SIGCSE '10 Proceedings of the 41st ACM technical symposium on Computer science education*, pp. 346–350, 2010.

[11] 松澤芳昭, 酒井三四郎. ビジュアル型言語とテキスト記述型言語の併用によるプログラミング入門教育の試みと成果. 研究報告コンピュータと教育（CE）, pp. 1–11, 2013.

[12] Tracey Booth and Simone Stumpf. End-user experiences of visual and textual programming environments for arduino. In *Lecture Notes in Computer Science*, pp. 25–39, 2013.

[13] Wanda Dann, Dennis Cosgrove, Don Slater, Dave Culyba, and Steve Cooper. Alice 3 to java. In *SIGCSE '12 Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pp. 141–146, 2012.

[14] Fraser McKay and Michael Klling. Predictive modelling for hci problems in novice program editors. In *BCS-HCI '13 Proceedings of the 27th International BCS Human Computer Interaction Conference*, 2013.

[15] Thomas W. Price and Tiffany Barnes. Comparing textual and block interfaces in a novice programming environment. In *ICER '15 Proceedings of the eleventh annual International Conference on International Computing Education Research*, pp. 91–99, October 2015.

対外発表

- 三重野 剛，三浦 元喜：プログラミング演習におけるブロック型インタフェースの導入と実践，大学 e ラーニング協議会 UeLA フォーラム（ポスター発表: 2pages），岩手，2018 年 2 月.