修士論文

プログラミング初学者のための
可視化システムによるプログラムの
動作確認の支援に関する研究

令和 3 年度修了

三重大学大学院工学研究科
博士前期課程　電気電子工学専攻

TRAN THANH TUNG

目次

第 1 章　はじめに

近年，情報社会の発展とともに，2020 年から小学校においてプログラミング学習が必須化されるなど，プログラミング学習の重要性が高まっている[1]．一般的なプログラミング学習方法として，演習型の授業が行われている．プログラム作成の演習において，学生の中にはプログラムを実行してみて動作の間違いことに気が付いてもその間違いを取り除けないものがいる．間違いを取り除くためには，実行途中のどこで・どうして正しい動作をしなくなったのかを知る必要がある．熟練者は簡単なプログラムの場合は頭の中でプログラムを 1 行ずつ実行することで変数や配列の値がどのように変化するのかを考えるが，初学者はこれを行うことが難しい．また，プログラミング学習において，プログラムの動作を理解する必要があるが，プログラムに分岐文や繰り返し文といった条件文の数が多いと，プログラミング初学者にとって困難である．その主な原因は，プログラムが複雑になると，プログラム実行時の変数の値の把握が難しくなるからである．

プログラム実行時の変数の値を知る方法として，一般的にデバッガが用いられる．しかし，デバッガは初学者向けに作られていないことが多く，初学者にとって利用法習得における負荷が大きいという問題がある．なぜならデバッガはプログラミング経験者のために開発されているからである．例えば，制御フローを可視化する機能の 1 つである「ブレークポイント」は，初学者が適切に設定するのは難しいと言われている[2]．そこで，プログラミング初学者のためにプログラム実行中の変数や配列の値を可視化し，プログラムの動作確認の支援をする可視化システムが開発されており，可視化システムはプログラミング学習に効果があると知られている[3, 4]．既存のものはプログラムのある行の実行直後の変数や配列の値を図的に表示するが，どの変数やどの配列の要素が変化したのか把握しにくい．可視化システムにより，プログラムのステップごとの変数や配列の値を可視化することで，学習者は各ステップでの変化が分かりやすくなる．しかし，プログラムの動作を理解するために，ステップごとの変化だけでなく，ブロックやメソッドを通した後，変数や配列の値がどのように変化するのかを把握することも重要である．既存の多くの可視化システムはステップごとの変化しか行われていないため，大局的なプログラムの変化の理解を支援していない．

本研究では，初学者向けの可視化システムを提案する．ステップごとの実行前後の比較の可視化，ブロックの実行前後の比較の可視化，という 2 種類の可視化を行い，学習者に動作理解の支援をする．ステップごとの実行前後の比較の可視化では，ステップごとの実行前と実行後を対比して表示し，変化した部分に色付けをし，変化した理由を明示する．ブロックの実行前後の比較の可視化では，ブロックの実行前後を対比して表示し，変化した部分に色付けをする．これらの変化を強調し，可視化することでプログラムがどのように動作し，変化するのか動作確認の支援をする．最後に構築したシステムを初学者に使用してもらい，アンケート調査により本システムの有効性を確かめた．

本論文の構成は以下のとおりである．2 章で本研究の背景について述べる．3 章ではデバッガとその問題点について述べる．4 章では先行研究について述べる．5 章では実装について述べる．6 章では実験について述べる．7 章で実験の結果と考察について述べる．8 章で授業での運用ついて述べる．9 章で授業で本システムで扱えないプログラムについて述べる．10 章で本論文のまとめを述べる．

第 2 章　研究背景

一般的なプログラミング学習方法として，演習型の授業方式が行われている．学習者はプログラムを作成する際に以下の工程を行う．

(1) コーディング
(2) コンパイル
(3) テスト
(4) デバッグ

学習者は，工程（3）のテストを行い，プログラムの動作結果が正しければプログラミング作業が完了し，正しくなければデバッグをする必要がある．デバッグするには図 1 に示すように，2 つの小工程が存在する．

(1) 間違えている位置の特定
(2) 間違えているコードの修正

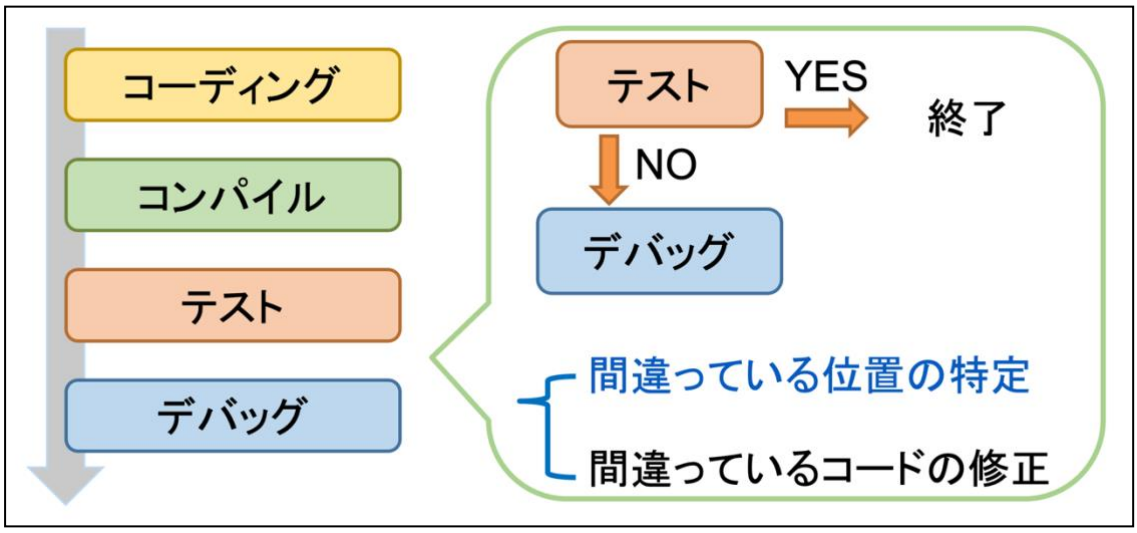

図 1　デバッグの手順

プログラミング初学者の中には自分が作成したプログラムの動作が間違っていても，間違っている位置を特定できないため，修正できない初学者が存在している．プログラミングにおいて，プログラムの動作を理解することは重要であるが，初学者にとってソースコードからプログラムの動作を把握することは難しい．その主な原因は，プログラムに分岐文や繰り返し文といった条件文の数が多いとプログラムが複雑になり，プログラム実行時の値の把握が難しくなるからである．

プログラム実行時の変数の値を知る方法として，一般的にデバッガが用いられる．しかし，デバッガは初学者向けに作られていないことが多く，初学者にとって利用法習得における負荷が大きいという問題がある．そこで，学習者に複雑な作業をさせずにプログラム実行中の変数や配列の値を可視化するシステムが有用であり，すでに多く存在している．既存のものはプログラムのある行の実行直後の変数や配列の値を図的に表示するが，どの変数やどの配列の要素が変化したのか把握しにくい．可視化システムにより，プログラムのステップごとの変数や配列の値を可視化することで，学習者は各ステップでの変化が分かりやすくなる．しかし，プログラムの動作を理解するために，ステップごとの変化だけでなく，ブロックやメソッドを通した後，変数や配列の値がどのように変化するのかを把握することも重要である．既存の多くの可視化システムはステップごとの変化しか行われていないため，大局的なプログラムの変化の理解を支援していない．

第 3 章　デバッガとその問題点

本章では，2 章で述べた代表的なデバッグを支援するシステム（デバッガ）を紹介する．また，以下に示すシステム以外にも，様々なシステムが存在している．

3.1　デバッグとは

デバッグとはソフトウェア開発上でプログラムの動作の誤り（バグ）を取り除く作業である．デバッグを行うためには，プログラム実行中の変数の値が正しいかを確認する必要がある．変数の値を確認する方法として，プログラムに直接デバッグコードを書く方法がある．例えば，Java 言語では，ある変数 `variable` の値を確認したい場合，`System.out.print(variable);`をソースコードに挿入し，変数 `variable` の値を出力させる．しかし，確認したい変数ごとにデバッグコードの挿入と再実行する必要があるため，効率的ではない．デバッグ作業を効率よく行うために，さまざまなツール（デバッガ）が開発されている．

3.2　デバッガとは

デバッガとはプログラムを特殊な環境で実行するソフトウェアであり，プログラムの実行を途中で停止し，その時点での全ての変数の値を表示する．デバッガを使う際，確認したい変数ごとにプログラムのソースコードの修正と再実行する必要がないため，プログラムの変数の確認を把握するために有用な機能である．デバッガはさまざまな統合開発環境（コンパイル・エディタ・デバッグ機能などがセットになったシステム）に組み込まれている．このような機能を持った統合開発環境としては Visual Studio Code[5]や NetBeans[6]などがある．この節では，Eclipse と DevTools について説明する．

3.2.1 Eclipse

Eclipse は，ソフトウェア開発を効率化するための統合開発環境であり，オープンソフトウェアとして公開されている[7]．Eclipse のデバッガ画面を図 2 に示す．Eclipse では，組み込まれているエディタの行番号の左側をダブルクリックすることでブレークポイントの設定と削除ができる．ブレークポイントを設定した後，デバッグボタンを押すとデバッガが起動し，プログラムの実行を中断させる．ブレークポイントで中断された位置での変数の値をエディタの右側で確認できる．デバッガの基本操作は図に示す①~④のボタンで行う．

① 次のブレークポイントまで実行する
② デバッガを終了させる
③ 1 行ずつ実行する（メソッドの呼び出し先まで深掘る）
④ 1 行ずつ実行する（クラス内において 1 行ずつ）

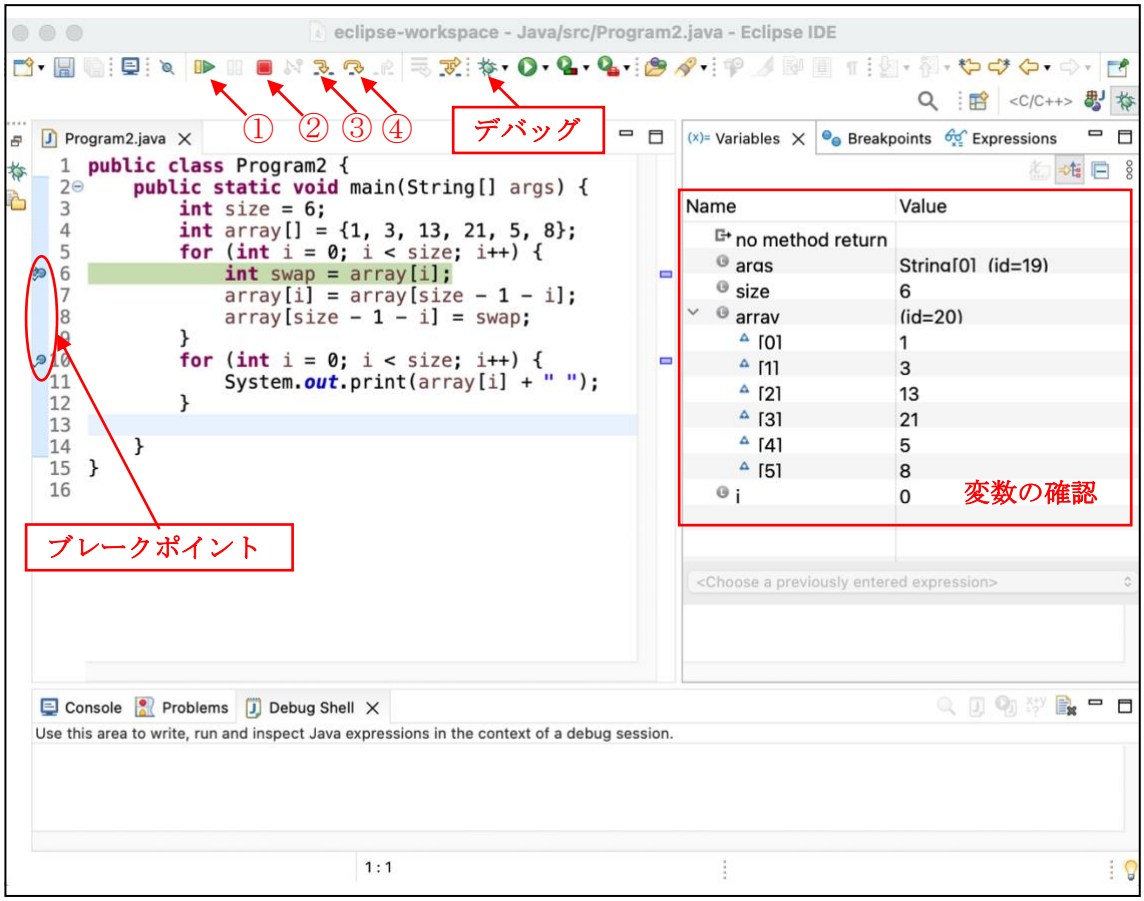

図 2　Eclipse のデバッガ画面

3.2.2 DevTools

DevTools は，Web ブラウザに付属している開発ツールである[8]．DevTools は HTML，CSS，JavaScript といった Web ページを構成するものの確認，編集とデバッグができる．DevTools のブレークポイント設定や基本操作は Eclipse と同じであるが，変数の値を確認するには，変数にマウスカーソルを合わせる方法（図 3）とコンソールに変数を入力する方法（図 4）がある．

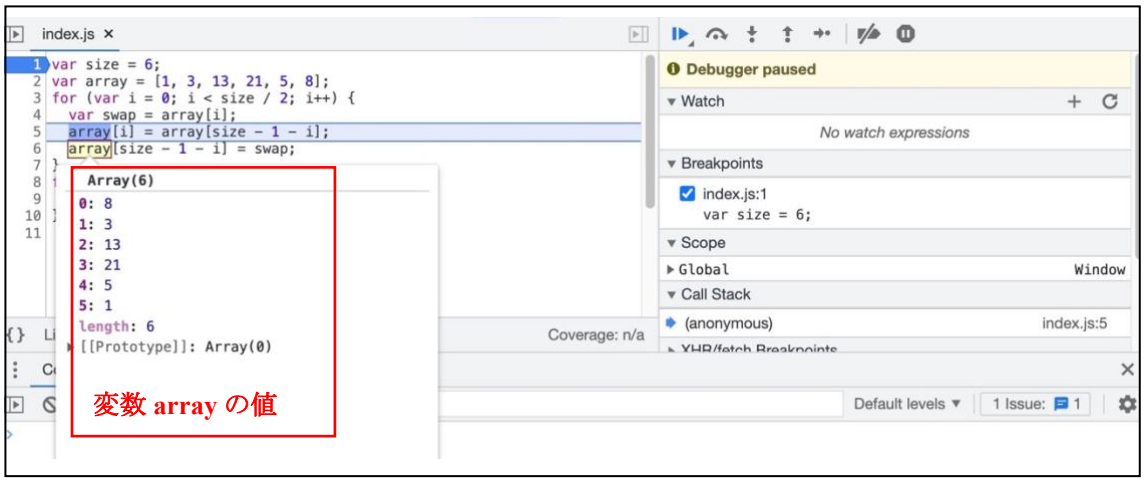

図 3　DevTools マウスカーソルを合わせる方法

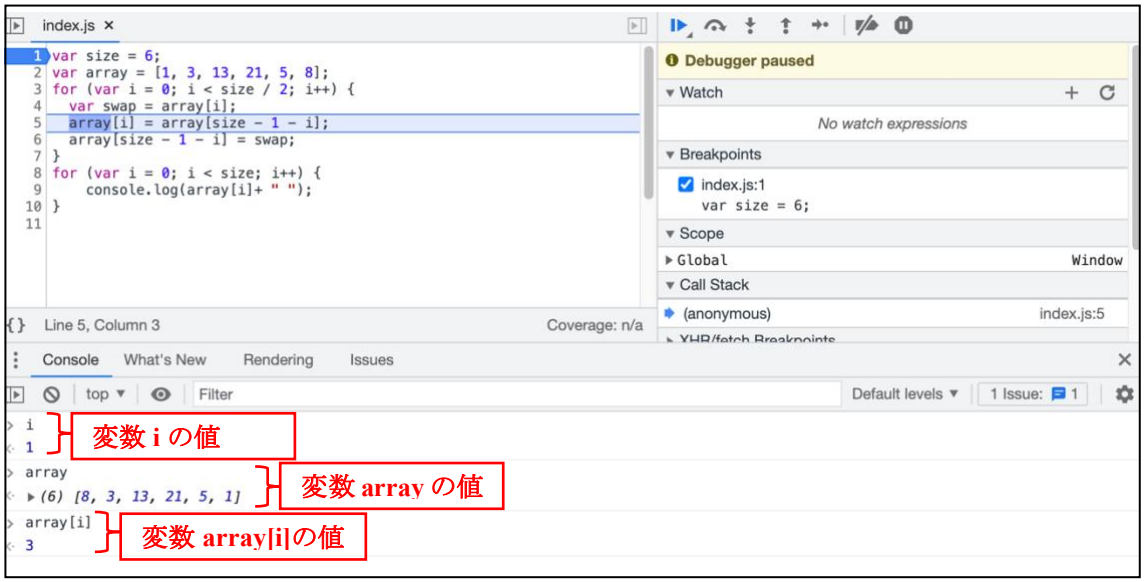

図 4　DevTools コンソールを使う方法

3.3　デバッガの問題点

この章ではデバッグ作業を効率的に行うため，いくつかのデバッガについて紹介した．しかし，プログラミング初学者にとって，これらのシステムには共通して以下の問題点がある．

(1) 統合開発環境の構築が必要

デバッガを使うためには，デバッガ機能を持った統合開発環境を整える必要があり，高機能な統合開発環境ほどその分 PC のスペックは高いレベルが要求される．パワーが足りない PC を利用すると，実行時間が長くなったり，処理が落ちたりすることがある．

(2) 機能が多く，使い方の学習の負荷が大きい

統合開発環境の主な機能はデバッグ，コンパイル，エディタ，ユーザーインターフェイス作成の 4 つである．1 つの機能を操るためには多くのボタンが存在し，どのボタンがどの機能するか学習負荷が大きい．

(3) ブレークポイントの設定の困難

デバッガを使う際，あらかじめ実行を中断させる位置を指定し，ブレークポイントを設定する．デバッガのブレークポイントを設定するためには，ある程度バグの位置を特定する必要がある．ブレークポイントを正しく活用しないと，そのブレークポイントは利用者に対してデバッグに有用な情報を与えらず，デバッグ効率の向上につながらない．

第 4 章　先行研究

3 章ではデバッガの問題点をあげた．プログラム実行中に変数の値を調べる方法として，デバッガ以外に可視化システムがある．可視化システムは利用者に余分な作業をさせないため，プログラミング初学者にとって有用なツールである．本章では，代表的な可視化システムを紹介する．また，以下に示す可視化システム以外にも，様々な可視化システムが存在している[9-13].

4.1　Online Python Tutor

Online Python Tutor は，学習者が書いたプログラムを，変数の値や配列の値を可視化することでプログラム学習支援を行うシステムである[14]．Online Python Tutor の可視化画面全体を図 5 に示す．ソースコード 1 行ごとの実行後の状態を可視化し，1 行ごとに進んだり戻ったりして状態を確認することができる．学習者は，Web ブラウザー上でプログラムを入力し，実行ボタンを押すことで可視化を行い，プログラム中で選択した行の実行後の変数の値や配列の値を見ることができる．しかし，一度に 1 つの実行後状態のみを表示するため，プログラム命令の実行前後で変数の値や配列の値の移り変わりを把握することが難しい．

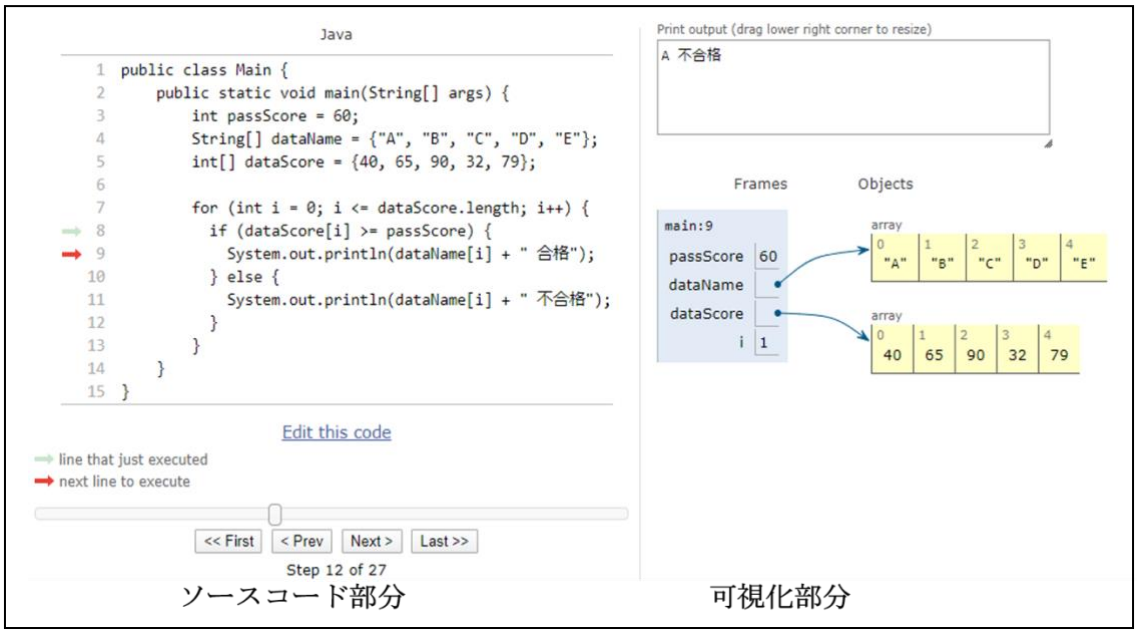

図 5　Online Python Tutor の可視化画面全体

4.2　伊藤の可視化システム

伊藤の可視化システムは，既存の可視化システムである Online Python Tutor を拡張したシステムである[15]. 伊藤の可視化システムはプログラムが 1 行ごとに実行するとき，前の状態と次の状態の比較ができるようになっている. 伊藤のシステムの可視化画面全体を図 6 に示す.

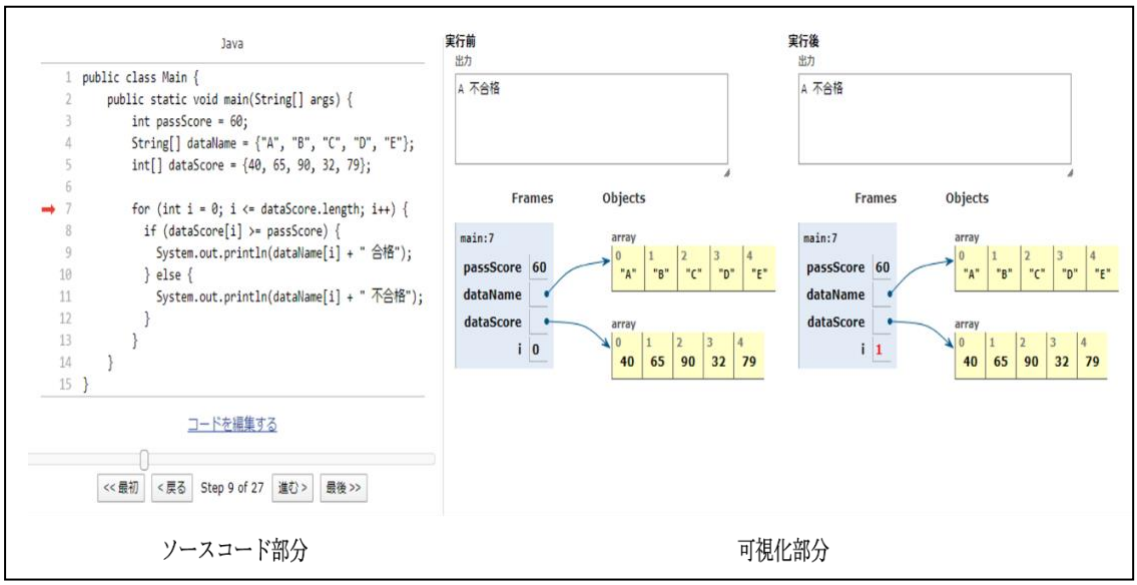

図 6　伊藤のシステムの可視化画面全体

伊藤の可視化システムは命令の実行前後で，変数の値や配列の値が変化した部分に，図 7 に示すように，色付けをして強調表示を行う．可視化部分に実行前後の状態を表示して変化を示すことで，プログラムのソースコード 1 行の実行前後の動作を確認できるようになっている.

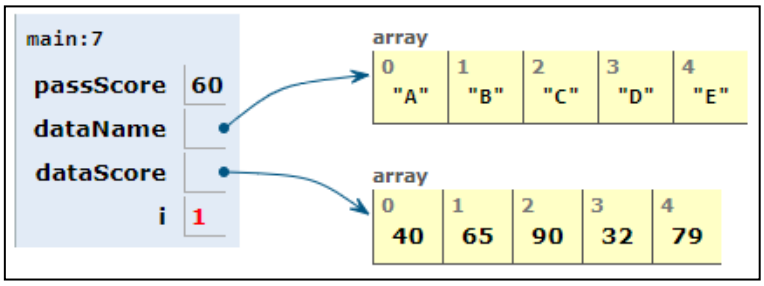

図 7　変化した部分の強調表示

伊藤が提案した可視化システムの機能を以下に示す.

(1) 学習者のプログラムで使用
(2) 図を用いて表現
(3) 図を自動生成
(4) 前の実行状態を表示
(5) 実行前後の状態を比較
(6) 変化部分を強調表示
(7) 変化した理由を表示
(8) ブロックの実行前後の状態を表示

伊藤は提案した機能のうち（1）から（6）までを実装した．伊藤が行った実験結果より，これらの機能は変数の値や配列の値の移り変わりを把握する支援に有効であることを確認した．伊藤が提案した（7）と（8）の機能については実装していない．（7）と（8）の機能の支援の必要性を以下に述べる．

- **変化した理由を表示**

学習者は条件文の評価式の内容や，プログラム中の計算の内容を読み取り間違えていることがある．そのため，評価式や計算式の詳細を示して，次のステップで変数の値が変化する理由の把握する支援が必要である．

- **ブロックの実行前後の状態を表示**

ソースコード 1 行の実行前後での可視化だけでは，プログラムの大きな視点から変数の値の変化を把握することが難しい．そのため，繰り返し文とメソッドを 1 つのブロックとしてみなし，ブロックの実行前後の変数の値を表示して支援する必要がある．特に，繰り返し文はブロックの前後の表示だけでは繰り返し文のループが進むごとの変数の値や変数の値の変化を把握することが難しい．そのため，1 回分のループを 1 ブロックとしてみなし，ブロック実行後の状態を表示して支援する必要がある．

第 5 章　実装

本研究は，Online Python Tutor と伊藤のシステムを拡張し，3 章で述べた伊藤が実装しなかった機能の実装を行う．本システムの可視化画面を図 8 に示す．画面の左側にはプログラムのソースコードを，右には変数の値や配列の可視化を表示する．

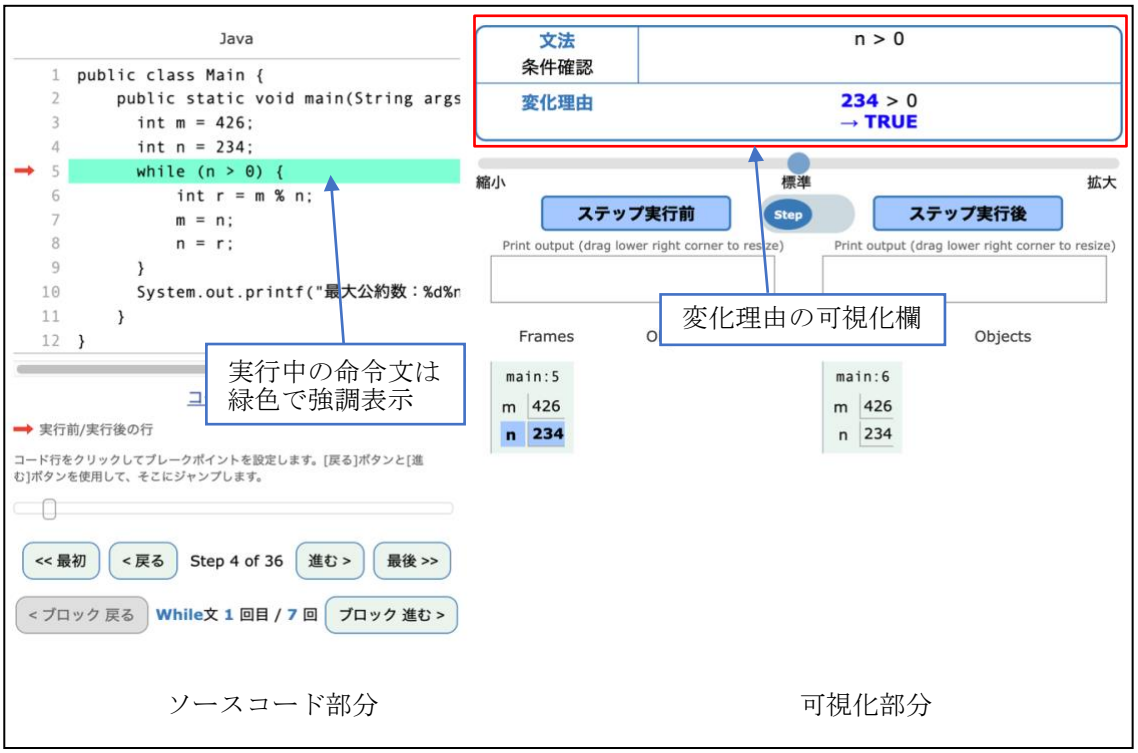

図 8　本システムの可視化画面全体

ソースコード部分では，プログラムが実行している行を認識しやすいために実行する行に緑色で強調表示する．可視化部分では，ステップ実行時に変数が変化する理由の可視化欄を追加した．

本研究は，以下の 2 種類の可視化手法を提案し，プログラミング初学者向けの可視化システムを実装する．

(1) ステップごと実行時の変化した理由の可視化

(2) ブロックごと実行時の実行前後の比較の可視化

5.1　変化した理由の可視化

プログラムのステップごとの実行前後の変化を可視化する．可視化を行っている命令に関係のある条件判定や繰り返し文の評価式の内容を表示し，その判定結果 TRUE，FALSE を表示する．同時に，計算式の内容を可視化部分に表示する．これらの表示により，学習者に変数・配列の値が変化した理由の把握を支援する．図 9・図 10 に示す実際の画面を 1 つの例として，ステップの実行時に変化する理由の可視化機能について説明する．

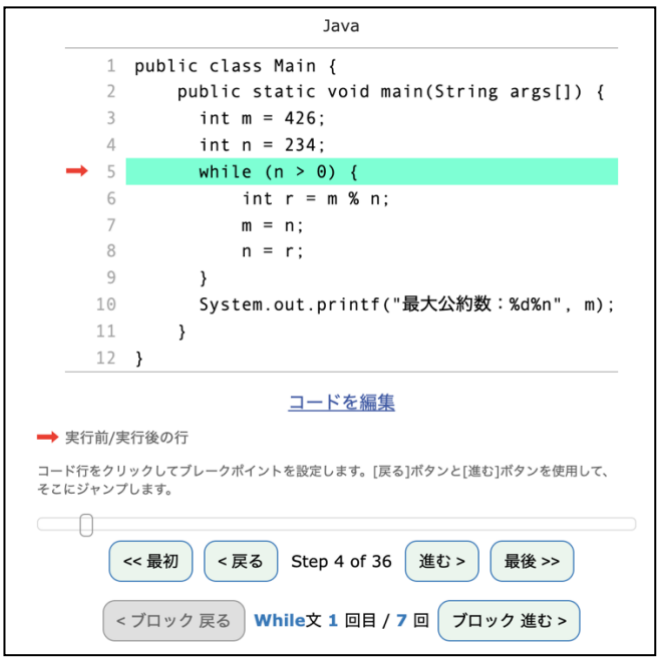

図 9　ステップ実行時のソースコード部分

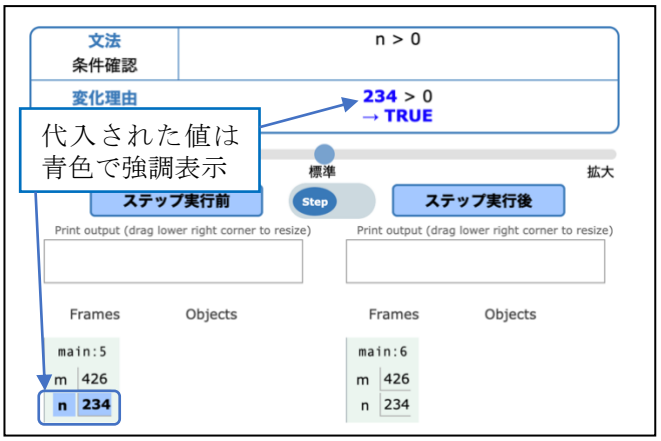

図 10　ステップ実行時の可視化部分

図 9 に示す最大公約数を求めるプログラムはステップ 4 で実行する while 文は n > 0 の条件式により，n の条件確認をする．この実行の可視化を行うと n = 234 を条件式に代入し，条件 n > 0 をチェックした後に TRUE の判定結果が表示される．この時の可視化は図 10 に示す.

5.2　ブロックごとの実行前後の比較の可視化

プログラムのブロック単位での実行前後の変化を可視化する.プログラムの繰り返し文やメソッドの動作は 1 ステップずつ実行を追いかけるだけでは理解しにくい時がある.ブロック単位で配列・変数の値の変化が可視化されることにより，繰り返しブロックの 1 ループごとの配列・変数の変化など，複数ステップを通した後，プログラムがどのように変化するのか分かりやすくなる.ブロックの実行前後を比較し，配列・変数の値が変更された箇所を強調表示することで，値がいつ変更されたのか分かりやすくなる．ブロックの実行前後の変化を表示することで，マクロな視点でプログラムの動作の把握を支援する.

本研究では，以下の 2 つをブロックの比較として可視化を行う.

(1) 繰り返しブロック（for，while，do…while）

(2) メソッド

5.2.1　繰り返しブロック

図 11 に示す 2 つの配列の要素を比較して，大きい方の値を結果配列に格納するプログラムを 1 つの例として，繰り返しブロックの実行前後の比較機能を説明する.

図 11 に示す繰り返しブロックを操作する部分より，for ループは全部 6 回があり，2 回目のループを実行していることが分かる．繰り返しブロックの可視化を操作するブロック戻るボタンとブロック進むボタンは繰り返しブロックを実行時のみ有効になる．これらのボタンを操作することで繰り返しブロックの各ループの実行前後の比較が確認できる.図 11 に示すソースコードのステップ 13 で実行する 13 行 for 繰り返しブロック実行前後の比較の可視化を図 12 に示す.

図 12 に示すステップ比較表示とブロック比較表示を切り替えるボタンは繰り返しブロックとメソッドを実行時のみ表示され，ブロック比較表示中でも，ステップ比較表示に切り替えることが可能である.

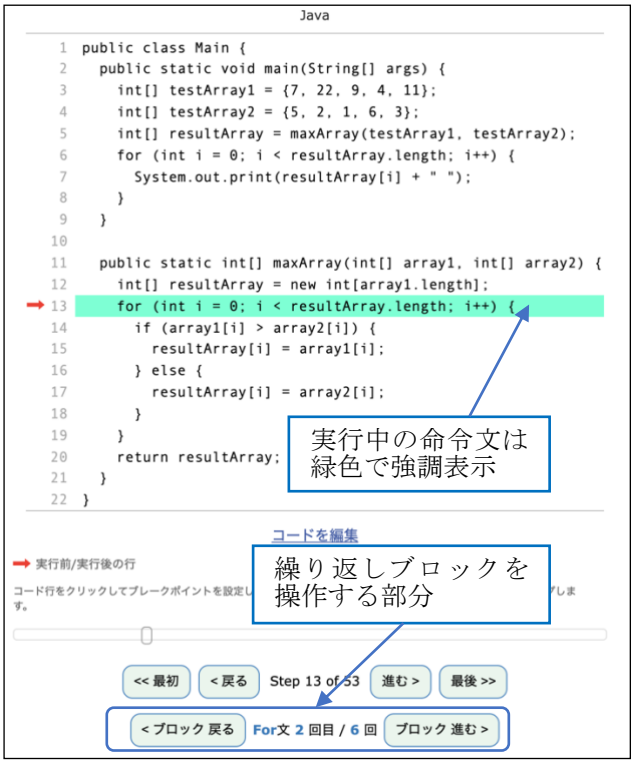

図 11　繰り返しブロック実行時のソースコード部分

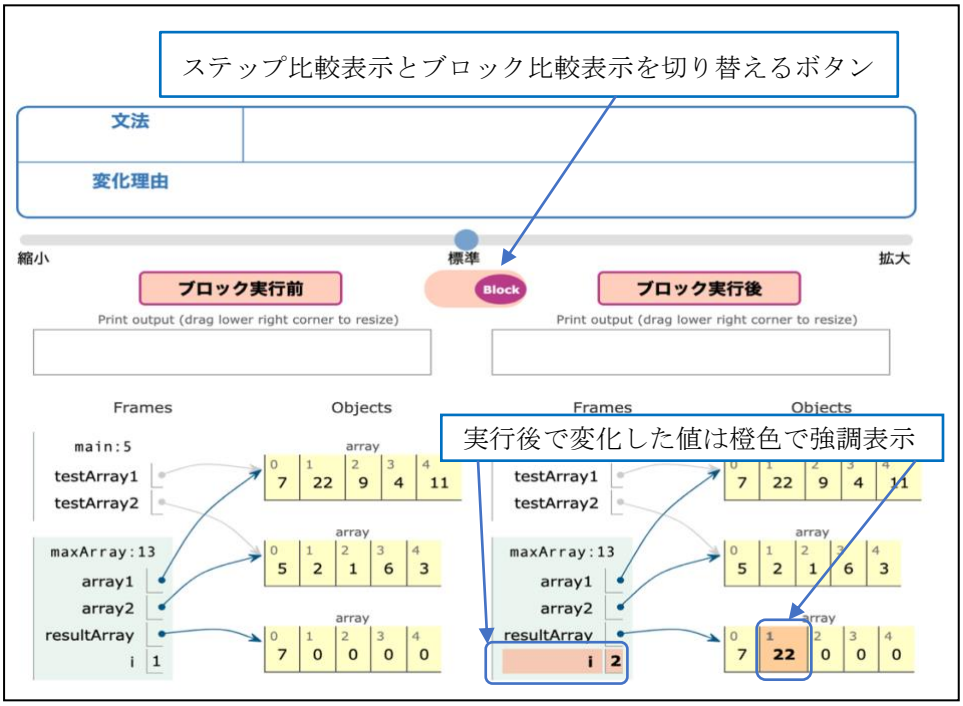

図 12　繰り返しブロック実行時の可視化部分

図 12 では，for 繰り返しブロック 2 回目ループを実行する前の値と，実行した後の値の両方を表示し，実行後で変化する値を橙色で強調表示する．この強調表示より，ステップ 13 で実行する for 文が 2 回目ループをした後，変数 i と配列 resultArray の位置 1 の値が変化することが一目で分かる．

この追跡作業を 3 章で述べた Eclipse のデバッガを用いると次のようになる．

① 13 行にブレークポイントを設定し，デバッグボタンを押すと変数 resultArray の最初の状態が確認できる（図 13）

② 「次のブレークポイントまで実行する」ボタンを押すと for 文の 2 回目ループを実行する前の状態が確認できる（図 14）

③ 「次のブレークポイントまで実行する」ボタンをもう一回押すと for 文の 2 回目ループを実行した後の状態が確認できる（図 15）

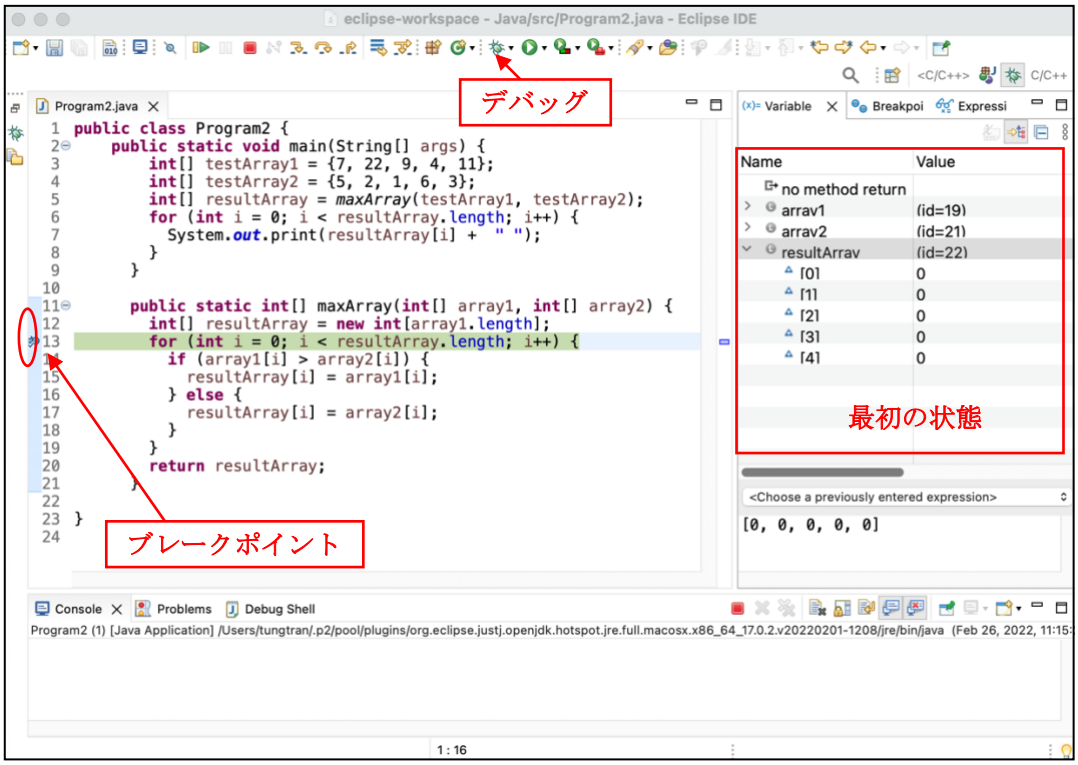

図 13　Eclipse のデバッガ画面（1 回目の中断）

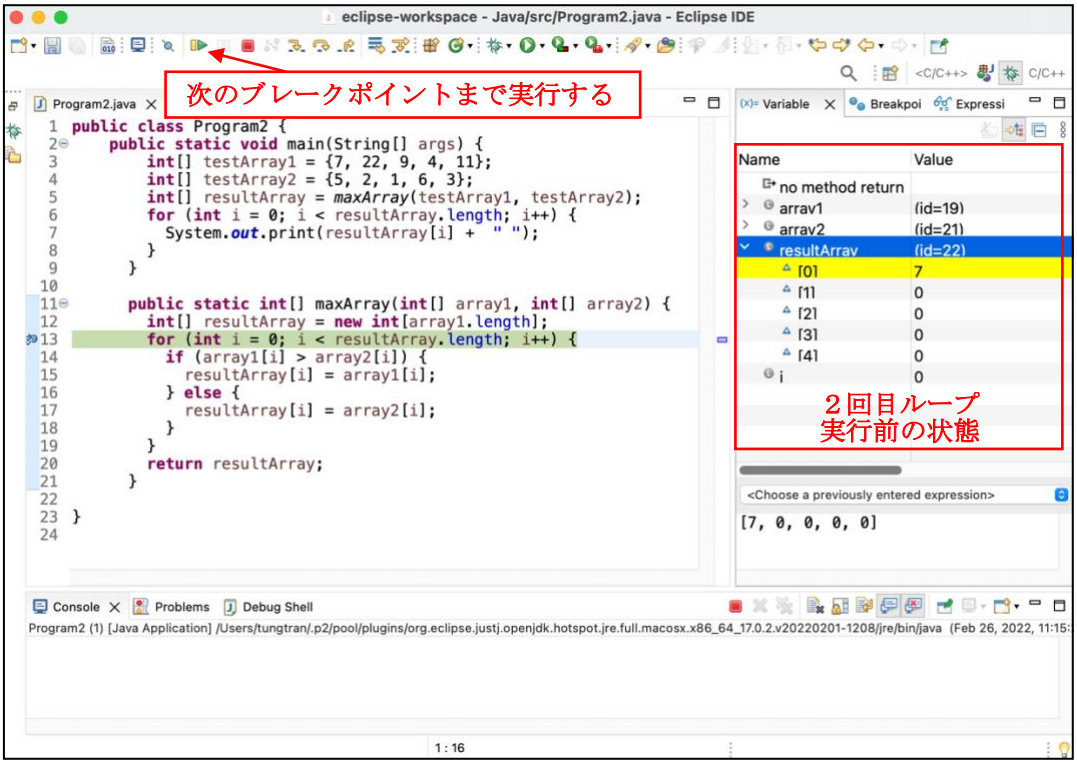

図 14　Eclipse のデバッガ画面（2 回目の中断）

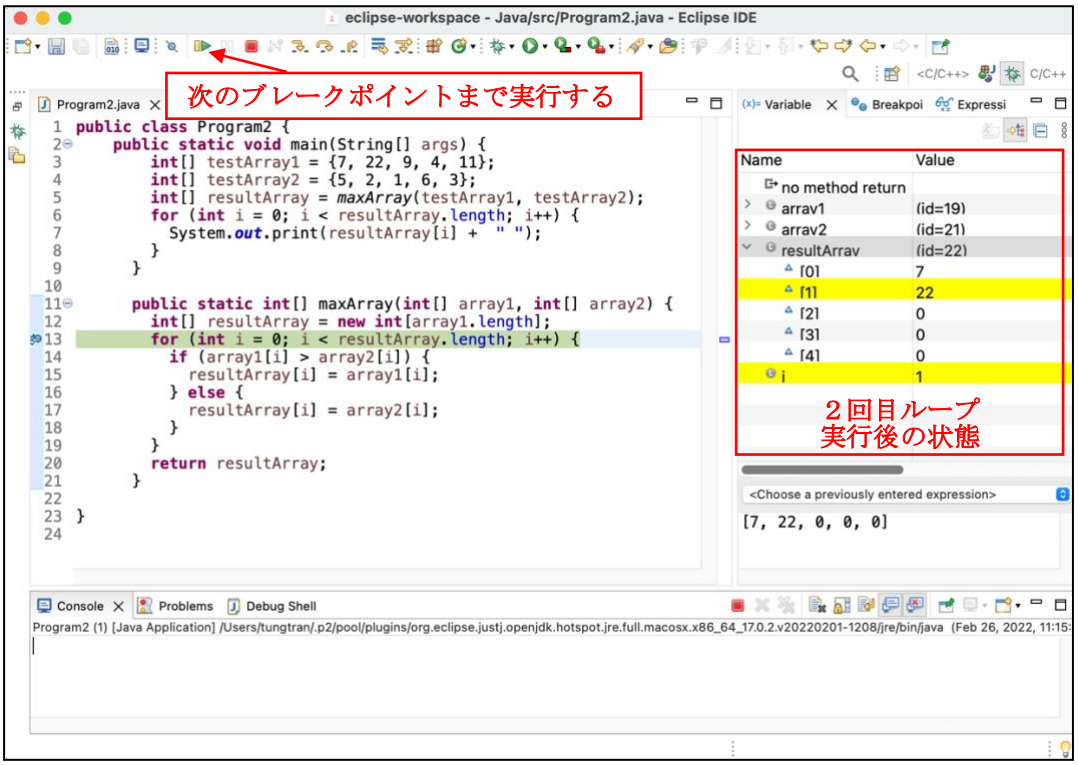

図 15　Eclipse のデバッガ画面（3 回目の中断）

Eclipse を使う際，13 行の for 文が 2 回目ループを実行した後，変数 resultArray がどのように変化するかを把握するには，図 14 と図 15 の状態を見比べる必要がある．また，デバッガのブレークポイントは次に進むことしかできないため，進み過ぎたときは最初からやり直す手間がかかる．

本システムは，実行前後の状態を同じ画面に表示しているため，変数の値の見比べが簡単になり，プログラムの変化を把握しやすい．また，従来のデバッガと違って，本システムでは，戻る機能があるため，変数の値の再確認や進み過ぎたときに役に立つ．

5.2.2　メソッド

図 16 に示す 2 つの配列の要素を比較して，大きい方の値を結果配列に格納するプログラムを 1 つの例として，メソッドの実行前後の比較機能を説明する．図 16 に示すソースコードのステップ 4 で実行する 5 行のメソッド maxArray の実行前後の比較の可視化を図 17 に示す．

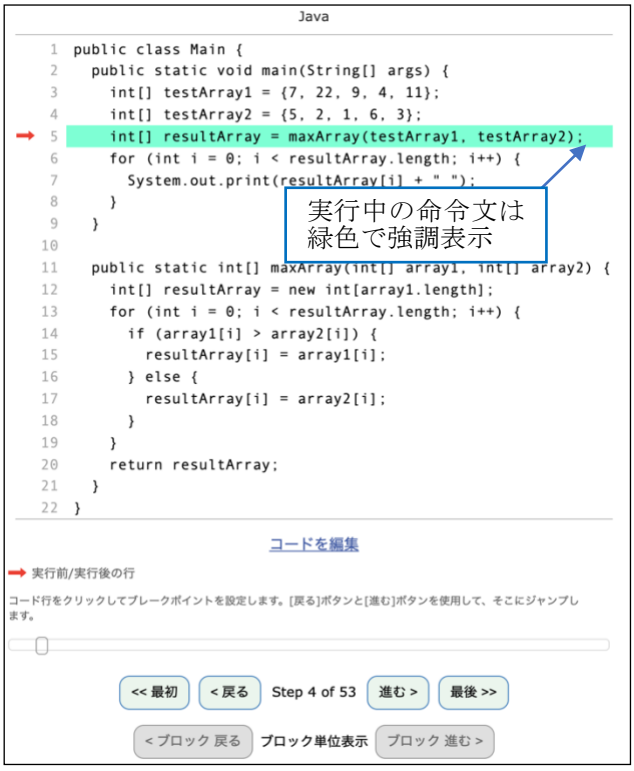

図 16　メソッド実行時のソースコード部分

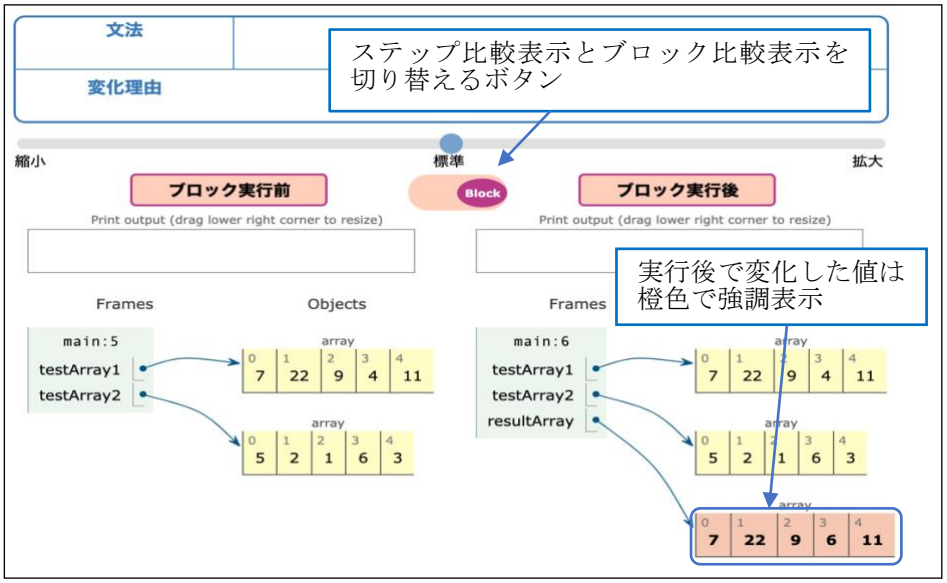

図 17　メソッド実行時の可視化部分

図 17 では，メソッド `maxArray` を実行する前後の値を表示し，実行後で変化する値を橙色で強調表示する．この強調表示より，メソッド `maxArray` を実行した後，配列 `resultArray` が新しく宣言されることが一目で分かる．

第 6 章　実験

本システムの提案により，学習者のプログラムの動作追跡に有効であるかを確認するため，以下の実験とアンケート調査を行った．

6.1　実験日と対象者

- 実験日　：2021 年 12 月 15 日
- 実験時間：60 分
- 対象者　：三重大学工学部電気電子工学科コース 1 年
  プログラミング言語 I の受講者
- 対象者数：97 人

6.2　実験方法

6.2.1 の 3 つのプログラムの誤りを学習者に修正してもらう．学習者は 6.2.2 の 3 つの作業を通して修正を行うという形式で実験した．3 つのプログラムは簡単な順番になっているが，学習者は作業中に次のプログラムに移ることを認める．作業中に次のプログラムに移る条件は以下の 2 つである．

(1) 現在のプログラムが修正できた
(2) 現在のプログラムが修正できず，ギブアップした

作業を始める際，ギブアップしたプログラムがあったら，ギブアップしたプログラムから修正することになる．実験後に学習者が回答してもらった報告結果とアンケート結果より，デバッグできた人数を解析し，従来のシステムと比較し，本システムの有効性を確認する．

6.2.1　プログラムリスト

(1) プログラム 1（簡単 – if 文の条件の誤り）

2 つの配列の要素の値を比較し，値が大きい方を結果の配列に格納するプログラム．修正させるプログラムを図 18 に，修正後の正解のプログラムを図 19 に示す．

```c
int main(void){
    int size = 5;
    int array1[] = {7, 22, 9, 4, 11};
    int array2[] = {5, 2, 1, 6, 3};
    int resultArray[5];
    for (int i = 0; i < size; i++) {
        if (array1[i] < array2[i]) {
            resultArray[i] = array1[i];
        } else {
            resultArray[i] = array2[i];
        }
    }
    for (int i = 0; i < size; i++) {
        printf("%d ", resultArray[i]);
    }
}
```

図 18　プログラム 1

```c
int main(void){
    int size = 5;
    int array1[] = {7, 22, 9, 4, 11};
    int array2[] = {5, 2, 1, 6, 3};
    int resultArray[5];
    for (int i = 0; i < size; i++) {
        if (array1[i] > array2[i]) {
            resultArray[i] = array1[i];
        } else {
            resultArray[i] = array2[i];
        }
    }
    for (int i = 0; i < size; i++) {
        printf("%d ", resultArray[i]);
    }
}
```

図 19　プログラム 1 の正解

(2) プログラム 2（難しい – for 文の条件式の誤り）

配列の要素の値を反転させるプログラム．修正させるプログラムを図 20 に，修正後の正解のプログラムを図 21 に示す．

```
1  int main(void) {
2    int size = 6;
3    int array[] = {1, 3, 13, 21, 5, 8};
4    for (int i = 0; i < size; i++) {
5      int swap = array[i];
6      array[i] = array[size - 1 - i];
7      array[size - 1 - i] = swap;
8    }
9    for (int i = 0; i < size; i++) {
10     printf("%d ", array[i]);
11   }
12 }
```

図 20　プログラム 2

```
1  int main(void) {
2    int size = 6;
3    int array[] = {1, 3, 13, 21, 5, 8};
4    for (int i = 0; i < size / 2; i++) {
5      int swap = array[i];
6      array[i] = array[size - 1 - i];
7      array[size - 1 - i] = swap;
8    }
9    for (int i = 0; i < size; i++) {
10     printf("%d ", array[i]);
11   }
12 }
```

図 21　プログラム 2 の正解

(3) プログラム 3（かなり難しい – 行数が多い）

大きい順に並べる 2 つの配列の要素を結合し，大きい順に並べる配列を作る．修正させるプログラムを図 22 に，修正後の正解のプログラムを図 23 に示す．

```c
int main() {
  int size1 = 4;
  int size2 = 4;
  int array1[] = {1, 4, 4, 13};
  int array2[] = {1, 2, 3, 9};
  int resultArray[8];

  int index1 = 0;
  int index2 = 0;
  int index = 0;
  while (index1 < size1 && index2 < size2) {
    if (array1[index1] < array2[index2]) {
      resultArray[index] = array1[index1];
      index1++;
    } else {
      resultArray[index] = array2[index2];
      index2++;
    }
  }
  while (index1 < size1) {
    resultArray[index] = array1[index1];
    index1++;
    index++;
  }
  while (index2 < size2) {
    resultArray[index] = array2[index2];
    index2++;
    index++;
  }
  for (int i = 0; i < size1 + size2; i++) {
    printf("%d ", resultArray[i]);
  }
}
```

図 22　プログラム 3

```c
int main() {
  int size1 = 4;
  int size2 = 4;
  int array1[] = {1, 4, 4, 13};
  int array2[] = {1, 2, 3, 9};
  int resultArray[8];

  int index1 = 0;
  int index2 = 0;
  int index = 0;
  while (index1 < size1 && index2 < size2) {
    if (array1[index1] < array2[index2]) {
      resultArray[index] = array1[index1];
      index1++;
    } else {
      resultArray[index] = array2[index2];
      index2++;
    }
    index++;
  }
  while (index1 < size1) {
    resultArray[index] = array1[index1];
    index1++;
    index++;
  }
  while (index2 < size2) {
    resultArray[index] = array2[index2];
    index2++;
    index++;
  }
  for (int i = 0; i < size1 + size2; i++) {
    printf("%d ", resultArray[i]);
  }
}
```

図 23　プログラム 3 の正解

6.2.2　作業リスト

本システムと従来のシステムの追跡効果を比較するため，実験を 3 つの作業に分け，各作業で使用したツールは以下の 3 つである．

作業1　可視化システムを使わずにプログラムを修正

- 可視化システムを使わずにプログラムを修正し，デバッグ結果を報告する作業である．
- 作業時間：20 分

作業2　従来の可視化システムを用いてプログラムを修正

- 従来システムである Online Python Tutor を用いてプログラムを修正し，デバッグ結果を報告する作業である．
- 作業時間：20 分

作業3　本システムを用いてプログラムを修正

- 本システムを用いてプログラムを修正し，デバッグ結果を報告する作業である．
- 作業時間：20 分

6.3　アンケート調査

本システムは学習者のプログラムの動作追跡に有効であるかを確認するため，使用後にアンケート調査を行った．アンケートの概要を以下に示す．

- 質問Ⅰ，Ⅱ
  - 本システムが提案した可視化機能が動作追跡するのに役に立ったかについての質問
- 質問Ⅲ，Ⅳ
  - 学習者のコメントや感想

第 7 章　実験結果と考察

6 章で行った実験の報告結果，アンケート結果を以下に示す.

7.1　各デバッグ作業の報告結果と考察

6.2.2 の各デバッグ作業の報告結果を図 24 に示す. 実験に 97 人が参加したが，修正したプログラムが提出されず，採点できないなどの理由を除いて，有効な報告件数は 68 件であった.

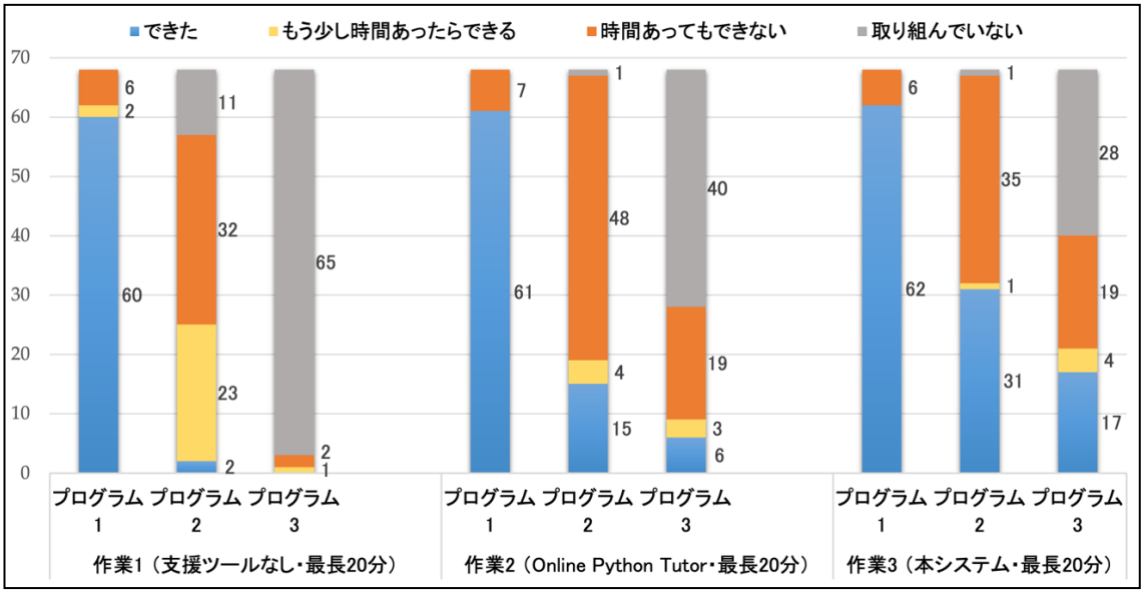

図 24　各デバッグ作業の報告結果

(1) 作業 1 の考察

作業 1（支援ツールなし）のグラフから，プログラム 1 が簡単だったため，可視化支援ツールを使用しなくても 60 人がデバッグでき，57 人がプログラム 2 のデバッグ作業に取り組んだ. プログラム 2 が修正できたのは 2 人で，23 人が「もう少し時間あったらできる」と答えた. 作業時間は 20 分で，プログラム 1 を修正した後，残りの時間は少なかったため，プログラム 2 の報告で「もう少し時間あったらできる」と答えた票数が他の作業と比べて最も多い結果となった.

(2) 作業 2 の考察

作業 2（Online Python Tutor）のグラフから，プログラム 1 が修正できた人数は 1 人が増え，プログラム 2 は 13 人が増えた．プログラム 3 では新たに 6 人が修正できた．プログラム 1 では，`if` 文の基本的な条件の間違いで，可視化ツールを使用しても効果が出ないということが分かった．基本的な文法が理解できない学習者に対しては，教師の特別な指導が必要となり，以降にプログラム 1 の報告結果を考察する必要はないと考えられる．プログラム 2 の報告では，「時間あってもできない」と答えた人は 48 人で，グラフ全体に最も多い．この原因として，作業 1（支援ツールなし）からプログラム 2 のデバッグ作業に取り組んだ学習者（55 人）が多くおり，作業 2 の 20 分を使っても修正できないためである．プログラム 3 については，行数が多く，初学者にとって複雑なプログラムである．作業 1 でプログラム 3 のデバッグ作業に取り組んだのは 3 人しかいなかったが，作業 2 の 20 分内に 6 人が修正できた．この結果より，従来システム（Online Python Tutor）はプログラムの動作理解の支援に有効であることを確認した．

(3) 作業 3 の考察

作業 3（本システム）のグラフから，プログラム 2 が修正できた人数は 16 人，プログラム 3 は 11 人が増えた．プログラム 2 の報告では，作業 2（Online Python Tutor）で 48 人が「時間あってもできない」と答えたが，本システムを使用した後，35 人に減った．プログラム 3 については，作業 2 で取り組んだのは 28 人で，修正できた人数は 6 人であったが，本システムを使用した後，17 人がプログラム 3 の修正ができ，11 人が増えた．この結果より，従来の可視化システム（Online Python Tutor）を使っても修正できない学習者が，本システムを使用すると修正できたことが分かった．また，本システムは動作追跡の支援ツールであり，本システムを使うと動作の間違いが必ずしも修正できるわけではないが，プログラムの動作追跡の支援に有効であることを確認した．

7.2　アンケート内容，アンケート結果と考察

7.2.1　質問Ⅰ，Ⅱの内容と結果

本システムが提案した変化理由の支援の有効を確認するため，学習者が本システムを使用した後，質問Ⅰに回答してもらった．質問Ⅰの内容を図 25 に，その結果を表 1 に示す．

質問Ⅰ
ステップごとを実行する際に，可視化部分の変化理由の支援は役立ちましたか？

1. 役に立った
2. どちらかといえば，役に立った
3. どちらかといえば，役に立たなかった
4. 役に立たなかった

図 25　質問Ⅰの内容

表 1　質問Ⅰに対する回答

	変化理由の可視化は動作追跡に役に立ったと感じたか				
選択肢	1	2	3	4	合計
票数（人）	39	29	12	6	86
百分率（%）	45	34	14	7	100

質問Ⅰのアンケート結果（表 1）より，「1（役に立った），2（どちらかといえば，役に立った）」の回答が 79%に達した．これより，本システムが提案した変数の値が変化する理由の可視化機能は，プログラムの動作追跡を支援することに有効である．

本システムが提案した 1 つのループの実行前後の比較の有効を確認するため，学習者が本システムを使用した後，質問Ⅱに回答してもらった．質問Ⅱの内容を図 26 に，その結果を表 2 に示す．

質問Ⅱ
繰り返し文を実行する際に，1 つのループの実行前後の比較は役立ちましたか？

1. 役に立った
2. どちらかといえば，役に立った
3. どちらかといえば，役に立たなかった
4. 役に立たなかった

図 26　質問Ⅱの内容

表 2　質問Ⅱに対する回答

	ブロックの可視化は動作追跡に役に立ったと感じたか				
選択肢	1	2	3	4	合計
票数（人）	39	23	18	6	86
百分率（%）	45	27	21	7	100

質問Ⅱのアンケート結果（表 2）より，「1（役に立った），2（どちらかといえば，役に立った)」の回答が 72%に達した．これより，本システムが提案したブロックの実行前後の比較の可視化機能は，プログラムの動作追跡を支援することに有効である．

7.2.2　質問Ⅲ，Ⅳの内容と結果

本システムの良い点を調べるため，使用後に，学習者に質問Ⅲに回答してもらった．質問Ⅲの内容を図 27 に，その結果を表 3 に示す．

> **質問Ⅲ**（任意）
> 質問Ⅰ，Ⅱで「1（役にたった），2（どちらかといえば，役に立った）」を選択した方に質問です．
>
> どのような点で役に立ったと感じましたか．

図 27　質問Ⅲの内容

表 3　質問Ⅲに対する回答

コメント	票数（人）
(1) 変化が分かりやすく，動作が理解できた	32
(2) 条件が成り立つかどうかを視覚的に確認できた	15
(3) 配列の入れ替えや繰り返し文の動作が分かりやすかった	13
(4) 日本語で書かれたのでわかりやすかった	2

質問Ⅲのアンケート結果（表 3）より，本システムに対して多くの肯定的なコメントを頂いた．表 3 の（1）～（3）のコメントは本システムが意図的に提案した機能に対するコメントであったが，(4) は機能以外のコメントであった．既存の可視化システムの多くは英語で書かれ，英語が苦手な学生には理解することが困難である．本システムの操作するボタンや構文の解説などは日本語に訳され，本システムの特徴である．

本システムの改善点を調べるため，使用後に，学習者に質問Ⅳに回答してもらった．質問Ⅳの内容を図 28 に，その結果を表 4 に示す．

質問Ⅳ（任意）
質問Ⅰ，Ⅱで「3（どちらかといえば，役に立たなかった），4（役に立たなかった）」を選択した方に質問です．

どのような点で役に立たないと感じましたか．

図 28　質問Ⅳの内容

表 4　質問Ⅳに対する回答

コメント	票数（人）
(1) 使い方が分からなかった	5
(2) 見てもプログラムが理解できなかった	5
(3) 見にくい	1

質問Ⅳのアンケート結果（表 4）より，本システムに対していくつかの否定的なコメントを頂いた．表 4 の（1）～（2）のコメントについては，本実験を行う当日に本システムの基本的な使い方のみ学習者に説明した．このため，本システムが提案した機能をどう使用すればプログラムが理解できるか分からない学習者がいた．本実験では，初めてこのシステムを使う人でも使用可能であるかを確認する意図があったので，妥協できる票数となっている．表 4 の（3）「見にくい」という意見があった．本システムは初学者を対象に実装し，システムが発展するたびに可視化内容の表示が増え，情報量が多くなり，システムの読みやすさの維持が必要となってくる．この問題の改善案として，変数の値を常に表示し，変化理由の表示といった発展した表示機能は講師や学習者がカスタムできるように改良すると考える．また，プログラミング初学者が本システムをデバッグ作業に有効的に使用し，プログラムのデバッグ作業の手順が理解するためには，講師の説明が必要となる．8 章で授業での運用の詳細を述べる．

7.3　全体の考察

7.1 と 7.2 の実験結果から，本システムによる変化理由とブロック実行前後の可視化の提案には妥当性があり，プログラムの動作追跡の支援に効果があると言える．また，本システムはプログラムの構文を具体的に解説してあるため，プログラミングの学習を始めた時点で使用すると一番有効であると考えられる．

本システムは実装済みで，プログラミング演習システム PROPEL（PROgramming Practice Easy for Learners）[16-18]に組み込んで，実際に三重大学工学部総合工学科電気電子工学コースのプログラミング言語 I の授業[19]で学習者が使用できるようになっている．実験後の「課題で行き詰まった時の対処法」のアンケート結果では，100 人のなかの 28 人がまた本システムを利用していることが分かった．

表 5　課題で行き詰まった時の対処法（2022/1/26　計：100 人）

順位	課題で行き詰まった時の対処法	前半（人）	後半（人）	票数（人）
1	ネットで検索する	39	25	64
2	教科書を確認する	30	33	63
3	友人，同級生に尋ねる	19	10	29
4	可視化ツールを使う	14	14	28
5	TA や講師に質問する	12	5	17
6	次回の授業で確認する	8	4	12
7	PROPEL の機能を確認する	5	3	8
8	その他	1	0	1

第 8 章　授業での運用

本システムはプログラムの動作追跡を支援する目的に実装した．プログラミング学習者だけでなく，プログラミング講師も本システムを授業の教材として使用することができる．プログラミング言語講師は，ソートのアルゴリズムなど配列の値の移り変わりが多く，複雑なプログラムを説明する時，教科書の図やフローチャートだけでは説明しきれないことがある[20]．本システムはソースコードを入力するだけで，プログラムの可視化を自動的に生成することができるため，講師がプログラムの動作を説明する時に利用できる．実際に三重大学大学院工学研究科の北英彦准教授がプログラミング演習授業で，本システムを使用し講義を行なっている．北英彦准教授が「講義の説明がしやすくなった」というコメントを頂いた．

6 章の実験で使用したプログラム 2（図 29）の修正過程を 1 つの例として，授業での運用仕方を説明する．

```
 1  int main(void) {
 2    int size = 6;
 3    int array[] = {1, 3, 13, 21, 5, 8};
 4    for (int i = 0; i < size; i++) {
 5      int swap = array[i];
 6      array[i] = array[size - 1 - i];
 7      array[size - 1 - i] = swap;
 8    }
 9    for (int i = 0; i < size; i++) {
10      printf("%d ", array[i]);
11    }
12  }
```

図 29　プログラム 2（配列を反転させるプログラム）

プログラム 2 の正しい結果は，配列 array の要素が反転され，「8 5 21 13 3 1」と出力するが，図 29 に示すプログラムを実行してみると「1 3 13 21 5 8」と出力し，配列 array の要素が反転されず，動作に間違いがあることが分かる．反転処理は，配列の先頭と最後の値の入れ替えから始まり，先頭の次の値と最後の 1 つ前の値を入れ替え，その処理を配列の中心に向かって入れ替えを行なっていく．この反転処理は，for 文により処理されるため，for 文の動作を確認する．

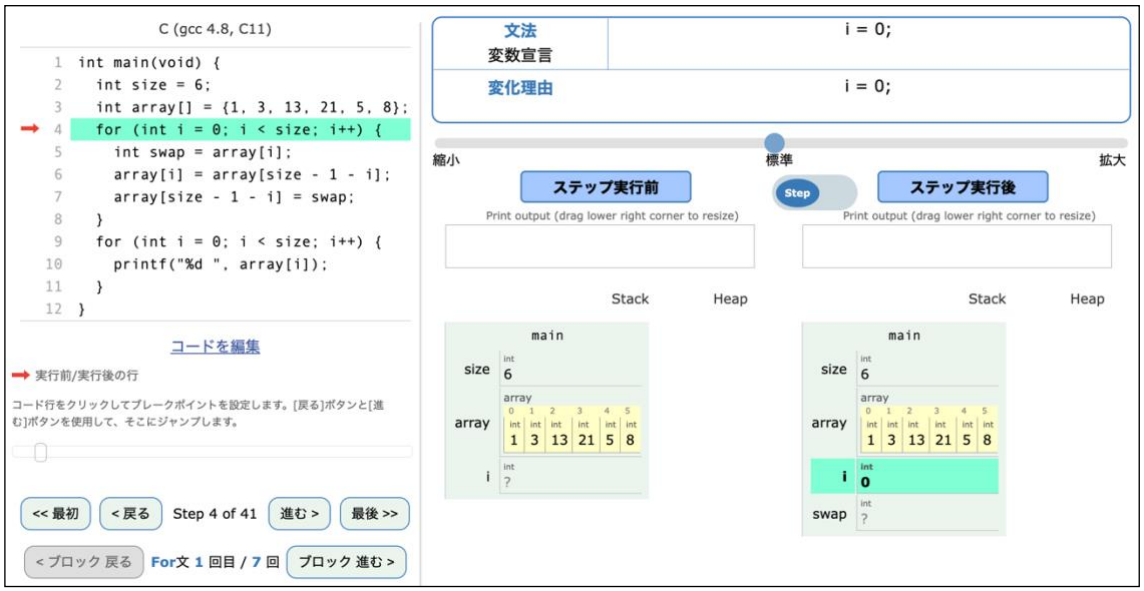

図 30　for ループ 1 回目（ステップごとの比較表示）

for 文の処理はステップ 4 から始まり，「ブロック進む」ボタンと「ステップ/ブロック比較表示の切り替え」ボタンが使用可能になる（for 文 1 回目ループしているため，「ブロック戻る」ボタンは使用不可）．この時のステップごとの比較は画面の右側に表示される（図 30）．「ステップ/ブロック比較表示の切り替え」ボタンを押し，ブロック比較表示に切り替える（図 31）．

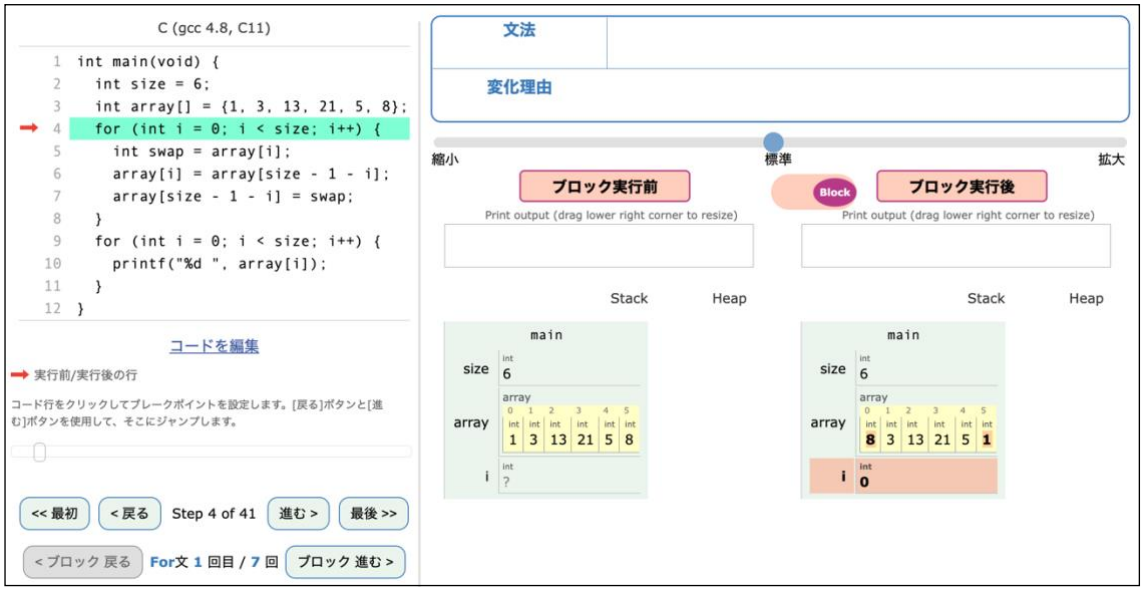

図 31　for ループ 1 回目（ブロックごとの比較表示）

図 31 の比較より，for 文 1 回目ループした後，配列の先頭の値と最後の値が入れ替わることが確認できる．次に「ブロック進む」ボタンを押し，for 文 2 回目ループの実行を確認する．

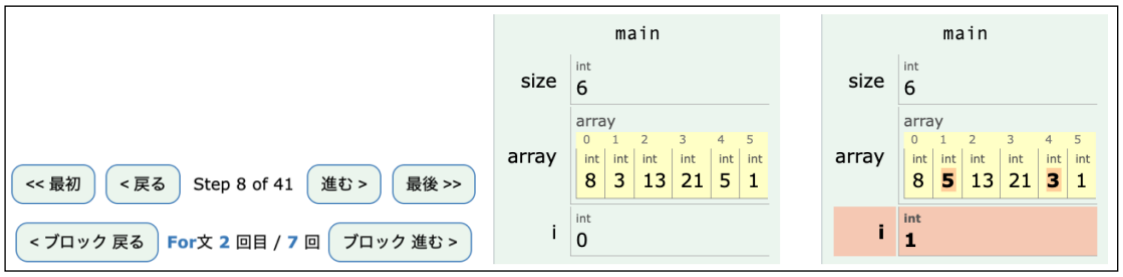

図 32　for ループ 2 回目（ブロックごとの比較表示）

図 32 の比較より，for 文 2 回目ループした後，配列の位置 1 の値と位置 4 の値が入れ替わることが確認できる．次に「ブロック進む」ボタンを押し，for 文 3 回目ループの実行を確認する．

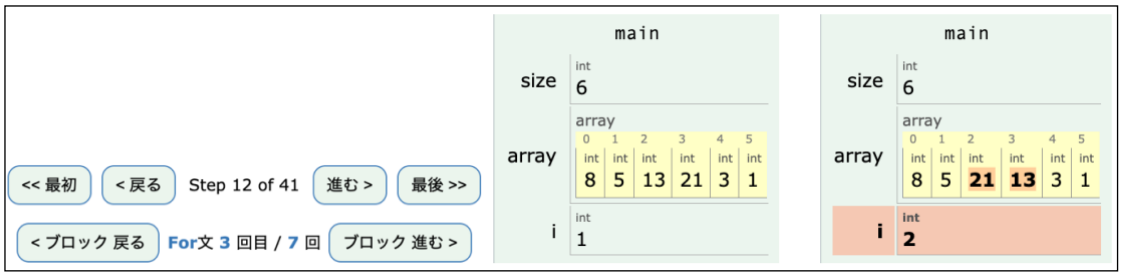

図 33　for ループ 3 回目（ブロックごとの比較表示）

図 33 の比較より，for 文 3 回目ループした後，配列の位置 2 の値と位置 3 の値が入れ替わることが確認できる．for 文 3 回目ループした後，配列 array の要素が正しく反転されることが確認できたが，for 文の処理はまだ残っているため，最後までの処理を確認していく．「ブロック進む」ボタンを押し，for 文 4 回目ループの実行を確認する．

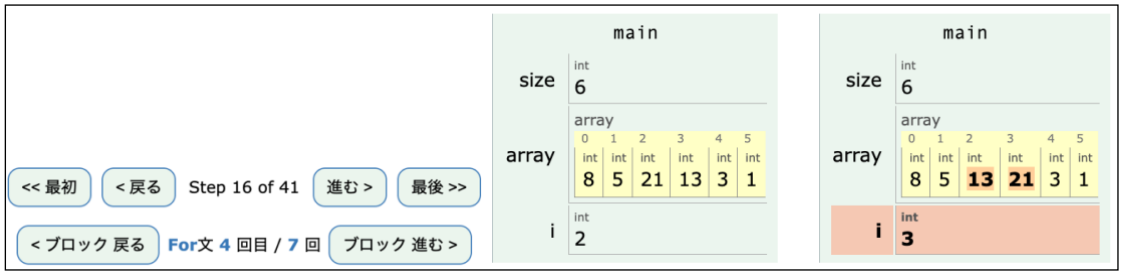

図 34　for ループ 4 回目（ブロックごとの比較表示）

図 34 の比較より，for 文 4 回目ループした後，配列の位置 2 の値と位置 3 の値が入れ替わることが確認できる．次に「ブロック進む」ボタンを押し，for 文 5 回目ループの実行を確認する．

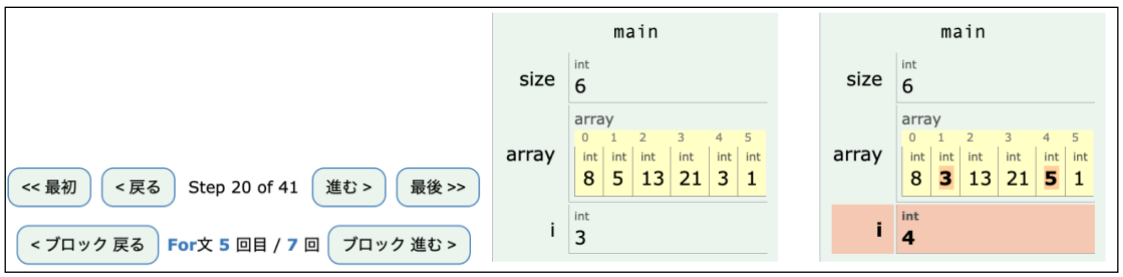

図 35　for ループ 5 回目（ブロックごとの比較表示）

図 35 の比較より，for 文 5 回目ループした後，配列の位置 1 の値と位置 4 の値が入れ替わることが確認できる．次に「ブロック進む」ボタンを押し，for 文 6 回目ループの実行を確認する．

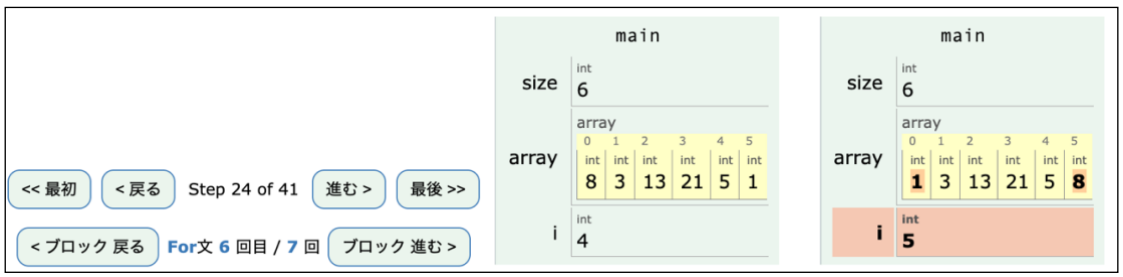

図 36　for ループ 6 回目（ブロックごとの比較表示）

図 36 の比較より，for 文 6 回目ループした後，配列の位置 0 の値と位置 5 の値が入れ替わることが確認できる．次に「ブロック進む」ボタンを押し，for 文 7 回目ループの実行を確認する．

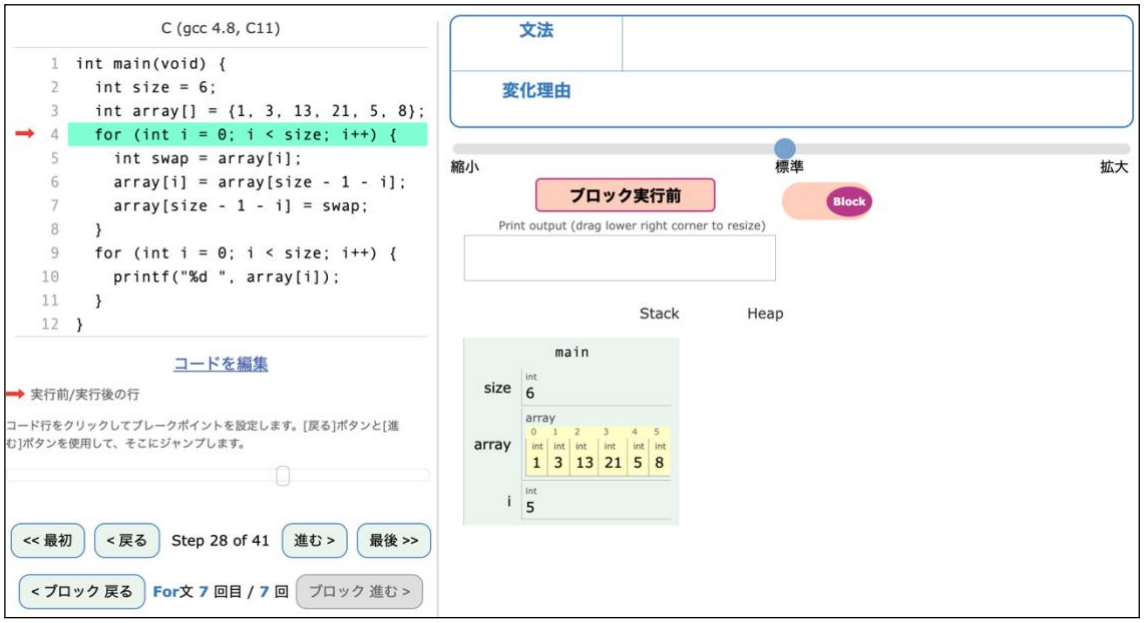

図 37　for ループ 7 回目（ブロックごとの比較表示）

図 37 では，for 文は全部 7 回ループがあるため，7 回目ループした後の状態が表示されない．また，「ブロック進む」ボタンが使用不可になる．図 37 の結果より，for 文の反転処理後，配列 array の要素が元に戻ることが確認できる．プログラミング初学者はこのように for 文の繰り返し条件の間違いに気づかず，配列の中心に向かって入れ替えるものを最後まで入れ替えてしまうケースが多い．

本システムは繰り返しループごとに実行前後の比較を 1 つの画面に表示するため，各ループの動作結果が可視化され，プログラムの動作説明で使用可能であり，学習者の理解度の向上にもつながる．

第 9 章　本システムで扱えないプログラム

本システムは，ユーザの Web ブラウザで可視化を行う前にサーバ側でプログラムを最後まで実行し，プログラムの実行結果（トレーサ）をユーザ側に返す．トレーサを生成するサーバと可視化部分の実装の都合上で扱えないプログラムを以下に示す．

- **大きなデータ構造を持つプログラム**

トレーサを生成するサーバの制限処理時間は 30 秒であり，大きなデータ構造を持つプログラムを実行する際，処理時間オーバーのエラー表示が現れ，可視化を行うことができない．

また，トレーサの最大格納は 1000 ステップまででであるため，無限ループやステップ数が 1000 を超えるプログラムは最後の実行までの可視化表示は不可能である．

- **ユーザに入力を要求するプログラム**

本システムでは，ユーザに入力を要求するプログラムは扱えない．入力が必要なプログラムは変数宣言に変えるなどの方法でプログラムを変更する必要がある．

- **コンパイルの前に前処理を行うプログラム**

本システムの可視化実装ではコンパイルの前に前処理を行うプログラムは扱えない（C 言語の `define` など）．トレーサから値が取得できないため，可視化の表示が正しく表示されないことがある．

第 10 章　まとめ

本研究では, プログラミング初学者が自身で作成したプログラムの動作追跡の支援することを目的として，Online Python Tutor と伊藤の可視化システムを拡張し，以下の機能を実装した.

- ステップごとの実行時に変化する理由の可視化
- ブロック実行前後の比較の可視化

そして実験結果より, 本システムがプログラムの動作追跡の支援に有効であることを確認した. また, 本システムが提案した 2 つの機能についてどちらも 70% を越える割合で高い評価が得られた.

参考文献

[1] 文部科学省 ： 小学校プログラミング教育の手引（第二版），p.4

[2] 吉村巧朗，亀井靖高，上野秀剛，門田暁人，松本健一:ブレークポイント使用履歴に基づくデバッグ行動の分析，電子情報通信学会技 術研究報告:信学技報，vol. 109，no. 307，pp. 85-90，2009

[3] Sassi Bentrad and Djamel Meslati:“Visual Programming and Program Visualization — Toward an Ideal Visual Software Engineering System −”，ACEEE International，vol. 1，no. 3，pp. 56‐62，2011

[4] Christine Alvarado，Briana Morrison，Barbara Ericson，Mark Guzdial，Brad Miller and David L. Ranum:“Performance and use evaluation of an electronic book for introductory Python programming”， Technical Report GT-IC-12-02， Georgia Institute of Technology，2012

[5] Kay Giza, Tobias Kahlert: “Visual Studio Code - Tips & Tricks Vol. 1”, Microsoft Deutschland GmbH，2016

[6] Netbeans project: https://netbeans.apache.org（2022 年 2 月参照）

[7] J. desRivieres，J. Wiegand :“Eclipse: A platform for integrating development tools”，IBM Systems Journal，vol. 43，no. 2，pp. 371‐383，2004

[8] Chrome DevTools project: https://developer.chrome.com/docs/devtools/（2022 年 2 月参照）

[9] 古宮誠一，今泉俊幸，橋浦弘明，松浦佐江子：プログラミング学習支援環境 AZUR−ブロック構造と関数動作の可視化による支援−，情報処理学会研究報告，ソフトウェア工学研究会報告，vol. 2014，no. 5，pp. 18，2014

[10] 中原進，紫合治：オブジェクト指向プログラムの動作の可視化，情報処理学会研究報告，ソフトウェア工学研究会報告，vol. 2010，no. 9，pp. 18，2010

[11] 中村亮太，西村知博，松浦敏雄：プログラミング入門教育用学習環境 PEN，情報処理学会研究報告，pp. 6571，2017

[12] 長慎也，甲斐宗徳，川合晶，日野孝昭，前島真一，筧捷彦：プログラミング環境 Nigari−初学者が Java を習うまでの案内役，情報処理学会論文誌：プログラミング，vol. 45，pp. 2546，2004

[13] 喜多義弘，片山徹郎，冨田重幸：Java プログラム読解支援のためのプログラム自動可 視化ツール Avis の実装と評価（ソフトウェアシステム），電子情報通信学会論文誌.D，情報・システム J95-D（4），pp. 855‐869，2012.

[14] Philip J. Guo : Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education，Proceeding of the 44th ACM Technical Symposium on Computer Science Education，pp. 579584，2013
[15] 伊藤 福晃，北 英彦：プログラミング演習における動作確認を支援するためのプログラム動作の可視化，2018PC カンファレンス論文集，pp. 29-32，2018
[16] 伊冨昌幸，北英彦，高瀬治彦，林照峯：コーディング状況に応じたアドバイスを可能にするプログラミング演習システムに関する研究，コンピュータ利用教育協議会，2010PC カンファレンス，2010
[17] 小島佑介，高橋功欣，北英彦：プログラミング演習における効率のよい指導のためのエラー早期指摘，コンピュータ利用教育協議会，2011PC カンファレンス，2011
[18] 小川正，西口大亮，北英彦：プログラミング演習における iPad などの携帯デバイスの利用による指導の円滑化, コンピュータ利用教育協議会, 2011PC カンファレンス，2011
[19] 三重大学ウェブシラバスプログラミング言語Ⅰ：https://syllabus.mie-u.ac.jp/?action=display&id=24263（2022 年 2 月参照）
[20] 太田剛：認知過程に考慮し，並び替えを目指す高校情報科のアルゴリズムプログラミング教育の開発と実践，情報教育シンポジウム　2021，pp. 85-92, 2021

実績一覧

国際会議発表

[a] Tran Thanh Tung, Hidehiko Kita, Haruhiko Takase : Program Visualization to Assist Debugging for Programming Beginners, Proceeding of The 11th International Symposium for Sustainability by Engineering at Mie University (Research Area C) (IS2EMU2021-C), pp.27-28, 2021

全国大会・支部大会

[b] TRAN THANH TUNG，北英彦：プログラムの各実行ステップの可視化によるプログラムの動作理解の支援, 2020PC カンファレンス論文集, pp.157-160, 2020

[c] TRAN THANH TUNG，北英彦，高瀬治彦：プログラムの動作の可視化によるプログラムの動作理解の支援, 2021PC カンファレンス論文集, pp.274-277, 2021

謝辞

本論文は，著者が三重大学大学院工学研究科博士前期課程に在籍中に行った研究をまとめたものである．本研究を進めるにあたり，懇切丁寧な御指導と御督励を賜った三重大学大学院工学研究科の北英彦准教授に深く感謝いたします．

また，貴重な時間をさいて本論文を査読して頂いた，三重大学大学院工学研究科の高瀬治彦教授，川中普晴准教授に深く感謝致します．

最後に，日頃熱心に討論していただいた計算機工学研究室の皆様方にお礼申し上げます．