Vol.2021-CE-159 No.12
2021/3/13

初学者のプロセス改善を目指した 漸進的プログラミング支援システムの開発と評価

吉田 賢志朗[1]　松澤 芳昭[1,a)]

概要：初学者のプロセスの改善を目指した漸進的プログラミング支援システム「トリップコード」を開発した．漸進的プログラミングとは，完成形までのステップを作成し，その都度保存・実行を行いながら，完成形へと近付けていくプログラミング手法である．トリップコードの特徴は，a) 保存時のコードの行数の推移が，折れ線グラフ形式で可視化されているため，保存の履歴を振り返ることが出来ること，b) 保存したバージョンの行き来が出来ること，の 2 点である．本システムを，社会情報学部の 1 年生のプログラミング導入授業で運用し，アンケート及びログの解析による効果の検証を行った．その結果，1) 約 4 割の学生が履歴表示機能を使って，自発的にグラフの意味を解釈し，漸進的に課題を解き進める足掛かりとしていたこと，2) 入れ子のような難易度の高い問題ほど，エラーが発生しやすいため，バージョントリップ機能の使用率が高かったこと，が明らかとなり，3) 難易度の高い問題の解答履歴グラフを見ることにより，その学生の習熟度を推定出来るという感触も得られた．

キーワード：初学者，漸進的プログラミング，プログラミング教育，プロセス改善，ステップ

1. はじめに

プログラミング導入教育の現場では，エラーの原因を特定することが出来ず，途方に暮れてしまう学生が散見される．彼らはエラーが発生した際，原因であると思われる箇所を予想して修正を行うが，編集した箇所が誤っている場合，更にエラーを重ねてしまう．それ故にエラーが複雑化し，自力で対処することが困難となってしまう．

楠ら [1] は，プログラミングの初学者が，文法やプログラミング言語特有の約束事を覚えるのに時間を費やしているため，アルゴリズムを十分に学習するまでの負担が大きいことを問題視している．アルゴリズムの学習不足が論理エラーを呼び，原因を特定することが出来ず，行き詰まってしまう．

我々は，一度に大量のコードを書かず，動作チェックをしながら少しずつ構築していくことが，重要ではないかと考えている．そうすることで，エラーが発生した際，原因だと考えられる範囲が小さくなり，特定がしやすくなるため，着実に問題を解き進めることが出来る．

更には，少しずつ解き進めることにより，アルゴリズムの設計にかかる負担を減らすことが出来ると考えられる．与えられた問題のアルゴリズムを，一度に全て設計することは，問題によっては困難である．それ故に，少しずつプログラムを実行してテストを行い，徐々にアルゴリズムを設計することで，誤りへの対処が早くなるだけでなく，負担を減らすことが出来る．

本研究では，完成形までのステップを作成し，保存・実行を行いながら進める，漸進的プログラミングという手法と，その支援システムを提案し，効果の測定を行った．

2. 先行研究

ソフトウェア開発において，実装が曖昧な箇所が出現した場合，複数種類の実装を試行・評価しながら開発を進めていくスタイルは，探索的プログラミングと定義されている．槇原ら [2] は，探索的プログラミングを促進するシステム「Pockets」を提案している．使用実験の結果，エラーが発生した際，その直前のプログラムを表示していることで，学習者が複数種類の実行を試し，探索的プログラミングを積極的に行うようになることを報告している．

初学者が漸進的に課題を解き進められない原因の 1 つとして，アルゴリズムの設計に関する知識・経験が，不足していることが挙げられる．松澤ら [3] は，学生に作業見積もりを作成させた際，作業項目の数や質に対し，差が顕著に見られたと報告している．

[1] 青山学院大学 社会情報学部
School of Social Informatics, Aoyama Gakuin University
[a)] matsuzawa@si.aoyama.ac.jp

Vol.2021-CE-159 No.12
2021/3/13

図 1 10 個の四角形

```
var t = createTurtle();
for (var i = 0; i < 4; i++) {
  t.fd(20);
  t.rt(90);
}
```

図 2 ステップ 1 のプログラム

```
var t = createTurtle();
for (var i = 0; i < 4; i++) {
  t.fd(20);
  t.rt(90);
}
t.up();
t.rt(90);
t.fd(25);
t.lt(90);
t.down();
```

図 3 ステップ 2 のプログラム

新開らは，アルゴリズムの正しさを確認せずに，プログラムを作成している学生がいることから，プロセスに着目したシステムを開発している [4]．システムを授業で使用した結果，システムを使用していない年度に比べて，問題を段階的に詳細化し，アルゴリズムを作成する力が伸びていると報告している [5]．

初学者がプログラミングを学ぶ際は，静止画よりも動画が適しているとされている．過程を動画収録するシステムは，安田ら [6] や，Bennedsen ら [7] などが挙げられる．それらのシステムを使用した学生は，熟練者のノウハウを見ることで，学習の助けになったと報告している．

TA や教師が，学習の遅れている学生をいち早く発見することも，プログラミング導入教育において重要である．井垣ら [8] は，受講生のコーディング履歴から，学習状況を可視化するシステム「C3PV」を開発し，講師・TA に使用させ，学習が遅れている学生を効率よく検出する方法を提案している．市村ら [9] は，エラー数や継続時間から助けを必要としている学生を検出するシステムを開発し，消えないエラーに困っている学生の手が挙がる前に発見出来たと報告している．

学習者の進度を，活動履歴から把握する試みもある．高橋ら [10] は，学習者の活動パターンを抽出する方法を提案している．Heinonen ら [11] は，学習者の過程をスナップショットで残し，合格者と不合格者の違いを分析していた．Toll ら [12] は，編集履歴や実行結果が，時系列で見ることが出来るシステムを開発している．

プログラミング学習を進める上でエラーが出た際は，トレースを行い，原因を特定する必要がある．江木ら [13] は，学習者が手間取っている箇所を質問から予測し，トレースを手助けするシステム「DESUS」を提案し，使用した学生

```
var t = createTurtle();
for (var j = 0; j < 10; j++) {
  for (var i = 0; i < 4; i++) {
    t.fd(20);
    t.rt(90);
  }
  t.up();
  t.rt(90);
  t.fd(30);
  t.lt(90);
  t.down();
}
```

図 4 ステップ 3 のプログラム

```
var t = createTurtle();
for (var j = 0; j < 10; j++) {
  for (var i = 0; i < 4; i++) {
    t.fd(20);
    t.rt(90);
  }
  t.up();
  t.rt(90);
  t.fd(30);
  t.down();
}
```

図 5 誤答のプログラム

が独自のトレースを試みていたことを報告している．

東本ら [14] は，プログラムを読ませる課題の提案と，誤りへのフィードバックを与えるシステムを開発し，実際に学生が使用して問題を解いた結果を報告している．テスト全てにおいて，トレース課題を行っていた学生の方が平均点が高かったことから，トレースする課題を解いている学生の方が，プログラムを構築する能力が身についていることを主張している．

3. 漸進的プログラミング

3.1 漸進的プログラミングとは

漸進的プログラミングとは，完成形までのステップを作成し，その都度保存・実行を行いながら，完成形へと近付けていくプログラミング手法である．漸進的プログラミングについて，**図 1** の例を用いて説明する．JavaScript で動作するタートルグラフィックスでの開発環境において，タートルを用いて 10 個の四角形を描画させる，という問題を考える．

図 1 の問題の場合，ステップに分けると，

ステップ 1 四角形を描く (**図 2**)
ステップ 2 次の四角形を描き始める場所まで移動する (**図 3**)
ステップ 3 その動作を 10 回繰り返す (**図 4**)

となる．このように，「四角形を描く」というような，完成形に至るまでの作業を分割したプロセスを，ステップと定義する．問題を解決するための小規模なステップを作成し，それらを 1 つ 1 つクリアしていくことにより，段階的に問題を解決する．これが，漸進的プログラミングである．

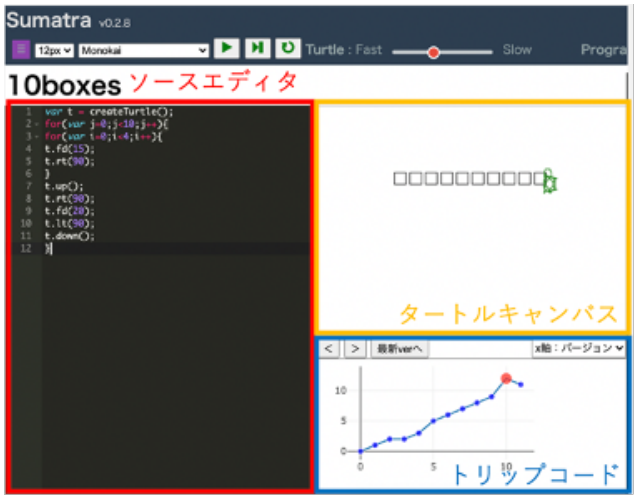

図 6 クラウド型 JavaScript 開発環境「Sumatra」

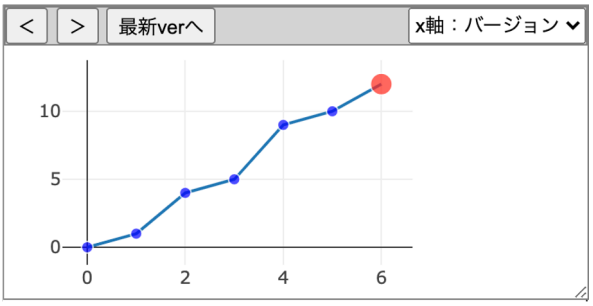

図 7 漸進的プログラミング支援システム「トリップコード」と解答履歴グラフ

3.2 漸進的プログラミングを行っていない場合のデバッグ

漸進的プログラミングを行っていない学生は，一度に 10 個の四角形を描画させようとする．実際に漸進的プログラミングを行っていない学生の回答例のソースコードを**図 5** に示す．この例では，図 4 の 10 行目にある，上を向かせるコードが抜け落ちており，正しい実行結果になっていない．

漸進的プログラミングを行っていない場合，ステップを作成していないため，どこまで出来ており，どこまで出来ていないかが判断出来ない．よって，3.1 漸進的プログラミングとは，で挙げた S2 をクリア出来ていないことに気付いておらず，7-10 行目に何か付け足したり，繰り返す箇所を変えたりと，勘で修正を加えてしまう．

4. 提案システム

本研究では，漸進的プログラミング教育支援システム「トリップコード」を提案する．

4.1 Sumatra の概要

Sumatra は，我々が独自に開発して，授業で導入している JavaScript の学習環境である．ブラウザで動作する Web アプリケーションタイプの開発環境であり，クラウドサーバにログインして利用する．ユーザが作成したファイルや操作記録はすべてクラウドサーバに保管される．2020 年度は，新型コロナウイルス拡大防止のため，実践授業はすべて遠隔授業で行われた．学生は，各自のコンピュータを用い，各作業スペースから Sumatra にアクセスし，プログラミング実習を行った．

Sumatra の外観を，**図 6** に示す．Sumatra にはタートルグラフィックスの動作環境が付属しており，平易な JavaScript を用いてタートルグラフィックスのプログラムを記述，実行することができる．ソースエディタとタートルキャンバスの 2 つに分かれており，ソースエディタにプログラムを記述し，実行ボタンを押すことで，結果がタートルキャンバスに表示される．

4.2 トリップコードの概要

4.2.1 2 つの機能

本研究で提案する「トリップコード」は，上記「Sumatra」に組み込まれた漸進的プログラミングの支援システムである．トリップコードの外観を，**図 7** に示す．支援対象として想定している学習者は，初めてプログラミング学習する者 (初学者) である．

トリップコードは，ソースコードの編集履歴を表示することによって，プロセスを可視化し，漸進的プログラミングを促進するシステムで，大きく以下の 2 つの機能を持つ．

機能 1．履歴表示機能 保存した回数，保存時のコードの行数の履歴が，折れ線グラフ形式で可視化されている．

機能 2．バージョントリップ機能 ユーザーが任意のタイミングで，保存したコードを表示することが出来る．

機能 1 で可視化される折れ線グラフの外観は，縦軸が保存時のプログラムの行数，横軸はバージョン (保存回数) である．このグラフを，解答履歴グラフと呼ぶ．Sumatra の実行ボタンを押すと，自動でプログラムが保存され，トリップコードのプロットが，右に 1 つ追加される．赤いプロットは，ソースエディタに表示されているプログラムのバージョンを表している．図 7 の場合，6 回目に保存したプログラムが．Sumatra のソースエディタに表示されている，ということである．

機能 2 は，プロットやボタンをクリックすることで，そのプロットが追加された時に保存していたプログラムが，ソースエディタに反映される機能である．「＜」のボタンを押すと，1 つ前のバージョンのソースコードを表示し，「＞」のボタンを押すと，1 つ後のバージョンのソースコードを表示する．「最新 ver へ」のボタンを押すと，最新のバージョンのソースコードを表示する．

4.2.2 2 つの機能の狙い

履歴表示機能は，ステップを作成しているか，振り返って確認するための機能である．グラフを見ることにより，コードの行数の推移が分かるため，急激に行数が増えている箇所を見つけた際，漸進的プログラミングを行っておらず，ステップを作成していないのではないか，という振り

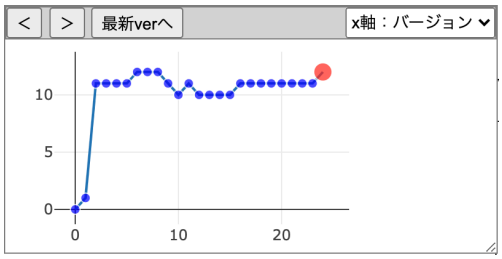

図 8 解答履歴グラフ (パターン 1)

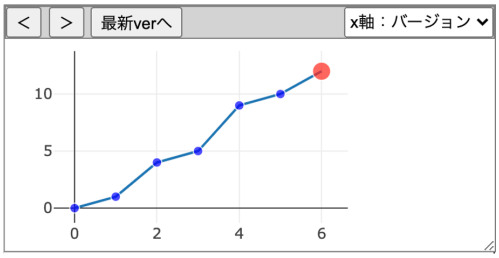

図 9 解答履歴グラフ (パターン 2)

返りが出来る．また，教師が学生の質問対応をする際，解き方の指導に繋げるための機能でもある．グラフを見て，急激に行数が増加している場合，エラーの原因が漸進的プログラミングを行っておらず，ステップを作成していないことである可能性が高いため，教師はその場で解き方の指導を行うことが出来る．

バージョントリップ機能は，学生が今まで作成したステップのコードや実行結果から，解き進めたプロセスを振り返るための機能である．学生はエラーを出力した際，以前のプログラムを表示し，実行することで，どの時点で追加したコードがエラーを引き起こしているのか，探して解き直す手助けをすることや，今まで保存したプログラムを見て，振り返ることが出来る．

4.3 使用シナリオ

本節では，使用シナリオについて説明する．ここでは，3.1 漸進的プログラミングとは，で扱った図 1 の 10 個の四角形を描画させる問題を例に挙げる．この問題に対して，トリップコードを用いて解くシナリオを，以降で説明する．

4.3.1 履歴表示機能の使用シナリオ

本節では，学生が履歴表示機能を使うシナリオについて説明する．

学生の A 君は，授業で課題を解いている．特に解けない問題も無く，順調に解き進めていたが，ある時壁にぶつかった．エラーが発生し，いくつか修正を加えてみたが，どうも原因が分からない．エラー箇所を探し続け，どうにか解くことが出来たが，かなりの苦戦を強いられた．一休みし，次の問題に進もうとしたが，ふとトリップコードを見ると，解答履歴グラフが**図 8** のようになっていた．後半はずっと平行線で，自分がいかに苦戦していたかが分かる．

そこで A 君は，なぜこんなにもエラーに苦戦したのかを考えた．その理由は，どうやらこの問題を解く際，一度に大量のプログラムを書いていることかもしれない．2 回目の保存で，プログラムが 10 行以上増えているからである．それ以前の問題では，多少最初にプログラムを多く書いたところで解けていたが，難易度の高い問題ではそうはいかない．

それから A 君は，その反省を活かし，少しずつ解くことを心がけるようになった．その結果，解答履歴グラフが**図 9** のように，緩やかになっていった．

図 10 タートル (パターン 1)

図 11 タートル (パターン 2)

4.3.2 バージョントリップ機能のシナリオ

本節では，学生がバージョントリップ機能を使うシナリオについて説明する．

B 君は，授業で図 1 の，四角形を 10 個描画させる問題を解いている．四角形を 10 個描画させるコード全てを書き，実行すると，**図 10** のようになった．タートルに四角形を描かせ，次の四角形を描く場所まで移動が出来，後は 10 回繰り返すだけだと考えていたが，どうやら間違っているらしい．

そこで，トリップコードの「<」ボタンを押して，1 つ前のステップのコードを確認した．そして，実行して結果を確認すると，**図 11** のようになっている．一見すると，次の四角形を描く場所まで移動出来ている．だがこの時点で，タートルは上を向いていなければいけないが，右を向いてしまっている．タートルを左に 90 度回転させて上を向かせ，もう一度，動作を 10 回繰り返すプログラムを書いたところ，正しい実行結果を出力することに成功した．

バージョントリップ機能を使用し，B 君はエラーに対処することが出来た．エラーの直前に保存をしていたことが，解き進める上で役に立ったので，B 君はより一層，少しずつ保存をしながら解き進めるようになった．

このようにして B 君は，トリップコードの手戻り機能を使うことにより，以前の実行結果が本当に正しいかを確認することによって，想定外の実行結果が出ても，対処することが出来た．それだけではなく，この手戻り機能に助けられた経験から，B 君はより一層漸進的プログラミングを意識して行うようになった．

表 1 授業内容

授業回	内容
第 1 回	基本的な命令
第 2 回	変数
第 3 回	条件分岐
第 4 回	繰り返し
第 5 回	入れ子構造
第 6 回	関数 (1)
第 7 回	関数 (2)
第 8 回	HTML の基礎
第 9 回	CSS の基礎
第 10 回	中間試験と自由課題 1
第 11 回	HTML と JavaScript の連携
第 12 回	Canvas の利用
第 13 回	タイマーとアニメーション
第 14 回	中間試験 (再) と自由課題 2
第 15 回	まとめ

5. 評価実験

5.1 研究目的

本研究の目的は，プログラミング入門教育において，漸進的プログラミングを支援するシステムであるトリップコードが，プロセス改善にどの程度寄与しているのかを示すことである．

5.2 分析対象

本研究の分析対象は，青山学院大学社会情報学部社会情報学科 1 年生の必修授業である，「コンピューティング実習」の受講者 230 名分のデータである．授業内容を，**表 1** に示す．受講者は皆，プログラミングの初学者である．

5.3 データの収集方法

本研究で分析するデータは，操作ログと，アンケートの 2 つである．操作ログは，トリップコードのボタン，もしくはプロットを押すことで，収集されるログであり，アンケートは，第 1 回・第 9 回・第 15 回の授業中にとったものである．

5.4 評価方法

5.4.1 使用状況の評価方法

トリップコードがどの問題でどれくらいの学生に使用されたのか，また，どの学生がどれくらいの問題で使用したのか，それぞれ使用率を計算した．計算方法は，

$$\text{問題別使用率} = \frac{\text{その問題で使用した学生数}}{\text{全学生数}} \quad (1)$$

$$\text{学生別使用率} = \frac{\text{その学生が使用した問題数}}{\text{全問題数}} \quad (2)$$

とする．

表 2 トリップコード使用率

使用率	平均	最大	最小
問題別使用率	6.3%	16.2%	2.1%
学生別使用率	6.3%	33.0%	0.0%

表 3 使用率の高い問題とその特徴

順位	問題の内容	特徴	問題別使用率
1	100box	入れ子	16.2%
2	キーボード	入れ子	14.1%
3	貝殻	入れ子	12.0%
4	ドラえもん	入れ子	10.6%
5	年齢計算	関数	9.9%

5.4.2 履歴表示機能の評価方法

履歴表示機能が想定通りの使われ方をされているのか確かめるため，アンケートで「解答履歴グラフを見て，一度に大量のコードを書いたことを反省し，次に繋げた経験はありますか？」と質問し，履歴表示機能に関して，コメントを収集した．

5.4.3 バージョントリップ機能の評価方法

バージョントリップ機能が想定通りの使われ方をされているのか，確かめるため，アンケートで調査を行った．質問内容は，

Q1 トリップコードを使って，解いた過程を確認しましたか？
Q2 トリップコードを使って，間違いを発見しましたか？
Q3 トリップコードを使って戻り，解き直しましたか？
Q4 トリップコードを使うことにより，少しずつ解くようになりましたか？

の 4 問である．

バージョントリップ機能に関しても，コメントを収集した．

6. 結果

6.1 トリップコードの使用状況と結果

問題別使用率・学生別使用率の，平均・最大・最小の値を，**表 2** に示す．問題別使用率は，最大 16.2%，最小は 2.1%であり，学生別使用率は，最大が 33.0%，最小は 0.0%，平均はどちらも 6.3%であった．トリップコードを 1 回以上使用した学生は，全体の 75.3%であった．

6.1.1 難しい問題で使用されているのか

トリップコードは，難しい問題で使用されているのか分析するため，問題別使用率の高い 5 問と，その問題の特徴・使用率を**表 3** に示す．使用率上位 4 つは入れ子の問題であり，使用率が 10%を超えている．

表 3 より，入れ子のように難易度の高い問題は，トリップコードの使用率が高いことが分かる．これらの問題は，他の問題に比べてエラーが発生しやすいため，エラー箇所を特定するため，使用率が上がっていると考えられる．

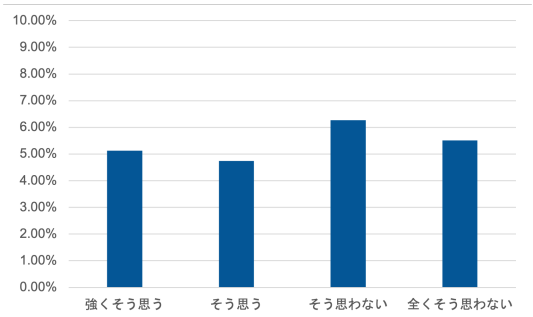

図 12 プログラミングの得意さごとの学生別使用率の平均

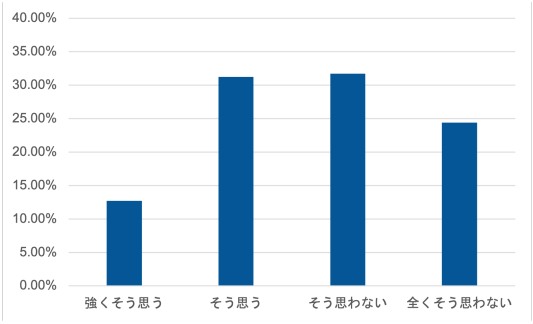

図 13 履歴表示機能に関する質問の結果

6.1.2 使用者はプログラミングが得意なのか

プログラミングが得意な学生がトリップコードを使用しているのか確かめるため，アンケートで学生に聞いた，「あなたはプログラミングが得意だと思いますか？」という質問に対する回答ごとに，学生別使用率の平均を出した．

そのグラフを，**図 12** に示す．4 つのどの回答者群を見ても，学生別使用率の平均値はおよそ 5%であった．この分析方法では，プログラミングの得意さと学生別使用率に関係は無く，得意な学生が使用している，という訳ではなかった．

6.1.3 使用者は成績が良いのか

使用者は成績が良いのかを確かめるため，トリップコードの学生別使用率と，中間試験の点数の相関を算出した．その結果，学生別使用率と中間試験の点数には，相関がなかった ($r \fallingdotseq 0.0004$)．

6.2 履歴表示機能に関する結果

結果のグラフを，**図 13** に示す．この質問に対し，「とてもそう思う」「そう思う」と回答した学生は，合計で 45%であった．

6.2.1 履歴表示機能に関する学生からの声

得られたコメントを，**表 4** に示す．

その結果，「スムーズにプログラミングができているときは右肩上がりのグラフになるが，微調整中や詰まっているときは横ばいのグラフになる．」のように，学生自らグラフの意味を考え，解釈している回答が 15 件ほど見つかった．

更には，「一度にたくさん書いてしまうと，チェックの際に剽窃を疑われやすくなると思って，こまめに実行しないとならなかったところが面倒であった．」のような，剽窃について記入されているものも散見された．解答履歴グラフから意味を解釈した学生が，剽窃を疑われないよう，漸進的プログラミングを行うようになっている．トリップコードは，漸進的プログラミングだけで無く，剽窃防止にも寄与していることが分かった．

表 4 履歴表示機能に対する受講者のコメント

受講者	コメント
A	最初のグラフのほうが傾きが大きかった．
B	最初は上がり，そのあとはほぼ水平に近くなっている．
C	1,2 回目で多めに書いている．
D	大幅にコードを変えると，急な右肩上がりになった．

表 5 バージョントリップ機能に対するアンケート結果

質問内容	そう思うと回答した割合
解いた過程を確認しましたか？	65.6%
間違いを発見しましたか？	51.5%
解き直しましたか？	65.6%
少しずつ解くようになりましたか？	62.5%

表 6 バージョントリップ機能に対する受講者のコメント

受講者	コメント
E	使用して間違いを発見できたので良かったです．
F	過去のプログラムと比較，戻ることができて良かった．
G	どこで間違えたのか把握しやすく，学習に役立った．
H	前に戻ることができ，問題点を見つけやすかったです．

6.3 バージョントリップ機能に関する結果

それぞれの質問で，「とてもそう思う」「そう思う」と回答した学生の割合を，**表 5** に示す．全質問で，5-6 割程度の学生が「そう思う」と回答し，1 つでも「そう思う」と回答している学生は，8 割以上存在した．

6.3.1 バージョントリップ機能に関する学生からの声

得られたコメントを，**表 6** に示す．

その結果，「プログラミングを戻せる機能はとても助かった．」など，この機能に対して好意的なコメントが，30 件ほど確認出来た．それらのコメントを残した学生は，積極的にバージョントリップ機能を使用していたと考えられる．

6.4 解答履歴グラフの分析

学生の中間試験の成績を，上位の A 群・中間の B 群・下位の C 群に分け，それぞれ 3 人の解答履歴グラフをまとめた．その図を，**図 14** に示す．

縦に学生が並んでおり，横に時系列順で問題が並んでいる．上 3 人が A 群，真ん中の 3 人が B 群，下の 3 人が C 群である．左側が授業序盤の問題であり，授業内容は，トリップコードのインストラクションを行った第 3 回から，タートルグラフィックス終了の第 7 回までである．

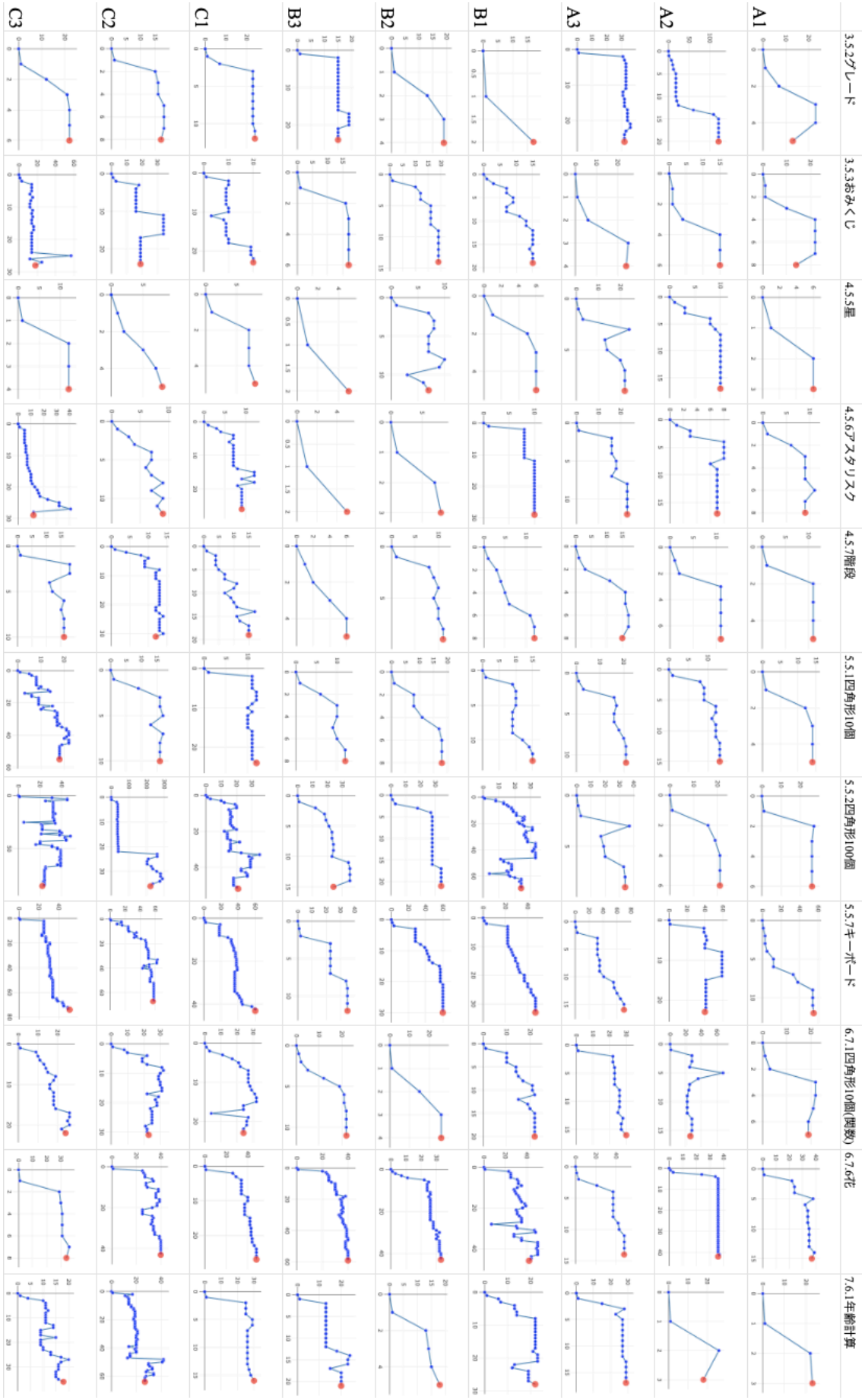

図 14 解答履歴グラフ一覧

解答履歴グラフを見比べると，難易度の易しい問題では，群での違いはほとんど見られず，形も様々である．

一方で，特に難易度の高い，四角形を 100 個描画させる問題の保存数を比べてみると，A 群はプロットが見えるほどしか保存していないが，C 群はグラフが上下を繰り返しており，プロットが見えないほど保存を繰り返している．

それ以外でも，比較的難易度の高い問題では，C 群の学生はグラフが上下している箇所が多く，苦戦していることが窺える．それに比べて A 群の学生は保存数が少ない，もしくは行数の変わらない保存が多く，上下している箇所はほとんど見られない．

比較的序盤の授業でも，C 群の学生は，難易度が少し上がると，苦戦している様子が窺える．おみくじ問題やアスタリスク問題では，A 群の学生は，数えるほどしか保存していないのに対し，C 群の学生は，プロットが多く，数えられないほど保存を繰り返していることが分かる．

躓く箇所は人それぞれであり，苦戦する度合いが上がるに連れて，保存回数が増えたり，行数の上下が増えていると予想される．特にグラフの形が上下している学生は，エラーの箇所を特定出来ず，書いては消してを繰り返していることが予想され，その問題にどの程度苦戦しているのか，ある程度は形から判断出来ると考えられる．

解答履歴グラフから，どの程度漸進的に解いているかを測ろうとしたが，容易ではないことも分かった．四角形を 10 個描画する問題→ 100 個描画する問題，のように，関連のある問題もあり，一概に最初に大量のコードを書いていることが，ステップを作成していないという判断に繋がるとは限らないからである．

少しずつプログラムの行数が増えていても，エラーが出続けており，ただ保存しているだけの学生も見られた．同じ行数の増加傾向でも，内容が大きく違う場合があり，少しずつ保存していることが一概に漸進的と言えないことが分かった．

6.5 解答履歴の分析

トリップコードのバージョントリップ機能を使用し，学生の解き進めている過程を確認した．18 人分を分析した結果，全員が漸進的に解いている問題から，誰も漸進的に解いていない問題まで，様々であった．中でも，キーボードを作成する問題 (**図 15**) は，18 人全てがステップを作成していた．この問題のみ，100 人分分析し，ステップを作成しているかどうか調べたところ，100 人中 97 人がステップを作成し，漸進的に解いていた．97 人の学生は全て，順番は違えど，

S1. 四角形を 30 個描く
S2. 大枠を描く
S3. 2 つのキーに線を描く

というステップで解いていた．このキーボード問題は難易度が高く，一度に全てをプログラムすることは難しいと判断した学生が，ステップを作成したと考えられる．

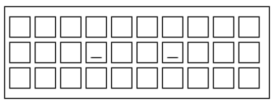

図 15 キーボードを描画させる問題

7. 考察

7.1 履歴表示機能はプロセス改善に役立ったのか

解答履歴グラフから，プロセスを反省し，次につなげたと回答した学生は，4 割以上存在した．解答履歴グラフの特徴に関しては，学生に言及していないため，自らグラフを見て，解釈していることになる．このことから，学生自ら，ステップを作成せずにプログラムを書いていることに気付き，反省することで，漸進的に解くきっかけとなっていたことが分かる．

剽窃を疑われないために，少しずつ保存を行っていた，と回答した学生も散見された．それらの学生は，剽窃を疑われないため，一度に大量のコードを書くことを避けている．想定していなかった使用方法であったが，彼らも自分なりにグラフの形を解釈した結果，そのようなコメントを残している．

この結果は，プロセス改善のシステムとして，評価出来る点である．

7.2 バージョントリップ機能はプロセス改善に役立ったのか

バージョントリップ機能を，想定通り使用していた学生は，8 割以上存在した．この機能は，解いた過程を確認・間違いを発見・解き直す際に使用されている．エラーが発生する前に保存していることにより，この機能の恩恵を受けることが出来るため，想定通り使用した 8 割の学生は，より一層漸進的に問題を解き進めるようになると考えられる．

問題別の使用率は，入れ子の問題で特に高くなっていた．入れ子の問題は，他の章に比べて難易度が高く，エラーの発生率が高い．それ故に，エラー前まで戻って解き直す学生が多いため，使用率が上がっていると考えられる．よってこのバージョントリップ機能は，プロセス改善よりも，デバッグの際に，エラーの箇所を特定するために使用する学生が多いことが考えられる．

このバージョントリップ機能は，プログラムを多く保存しているほど，得られるメリットは大きい．そのため，より機能の恩恵を受けるため，少しずつ保存や実行を行うようになっている，という結果が出ることが望ましい．

7.3 解答履歴グラフから得られた情報と考察

解答履歴グラフから，段々と行数が増えている「順調な

パターン」，行数が増加しないまま，保存数が増えている「少し苦戦しているパターン」，行数が上下し，かつ保存回数がとても多い「かなり苦戦しているパターン」など，いくつかパターン分けが出来た．だが，その学生が漸進的に解き進めているかどうかの判断はまだ出来ておらず，問題ごとにステップの大きさも異なるため，更なる調査が必要である．

現段階では，その問題に苦戦しているか否か，解答履歴グラフを見ると分かるという結果になっている．今後は，コンパイルエラーや実行時エラーが起こった際，プロットの色を変更したり，問題ごとの傾向や，学生ごとの傾向を調べることによって，ステップ作成の有無と解答履歴グラフの関係が分かるのではないか．

7.4 漸進的プログラミングに関しての考察

100 人中，9 割以上の学生が，キーボードの問題 (図 15) を漸進的に解いていることから，難易度が高く，一度に解けないと判断した際，ステップを作成して漸進的に解き進めていることが考えられる．しかしそれは，その他の問題でも，漸進的に解き進める事は出来るが，一度に解けると判断し，漸進的に解くという選択をしていない可能性があるとも言える．難易度の高い問題に対して，漸進的に解き進めない可能性もあるため，実力を過信している学生ほど，そのような事態に陥ってしまう可能性がある．

8. まとめ

履歴表示機能・バージョントリップ機能がある「トリップコード」を開発した．初学者約 200 人に，実際の授業で使用してもらい，漸進的プログラミングを支援することが出来るか，実験した．アンケート・ログを分析し，1) 約 4 割の学生が履歴表示機能を使って，自発的にグラフの意味を解釈し，漸進的に課題を解き進める足掛かりとしていたこと，2) 入れ子のような難易度の高い問題ほど，エラーが発生しやすいため，バージョントリップ機能の使用率が高かったことが明らかとなり，3) 難易度の高い問題の解答履歴グラフを見ることにより，その学生の習熟度を推定出来るという感触も得られた．

参考文献

[1] 楠　房子，宮内　新，小沢慎治：アルゴリズムスタイルを重視した情報処理教育，電子情報通信学会論文誌 A, Vol. 75, No. 2, pp. 441–448 (1992).
[2] 槇原絵里奈，藤原賢二，井垣　宏，吉田則裕，飯田　元：初学者向けプログラミング演習のための探索的プログラミング支援環境 Pockets の提案，情報処理学会論文誌, Vol. 57, No. 1, pp. 236–247 (2016).
[3] 松澤芳昭，岡田　健，酒井三四郎：Programming Process Visualizer：プログラミングプロセス学習を可能にするプロセス観察ツールの提案，情報教育シンポジウム 2012 論文集，No. 4, pp. 257–264 (2012).
[4] 新開純子，炭谷真也：プロセスを重視したプログラミング教育支援システムの開発，日本教育工学会論文誌，Vol. 31, No. Suppl., pp. 45–48 (2008).
[5] 新開純子，宮地　功：プログラミング学習支援システムを用いた入門教育の実践，日本教育工学会論文誌，Vol. 33, No. Suppl., pp. 5–8 (2009).
[6] 安田　光，井上亮文，市村　哲：学生とティーチングアシスタント間でトラブル解決過程を共有できるプログラミング演習支援システム，情報処理学会論文誌，Vol. 53, No. 1, pp. 81–89 (2012).
[7] Bennedsen, J. and Caspersen, M. E.: Revealing the programming process, *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pp. 186–190 (2005).
[8] 井垣　宏，齊藤　俊，井上亮文，中村亮太，楠本真二：プログラミング演習における進捗状況把握のためのコーディング過程可視化システム C3PV の提案，情報処理学会論文誌，Vol. 54, No. 1, pp. 330–339 (2013).
[9] 市村　哲，梶並知記，平野洋行：プログラミング演習授業における学習状況把握支援の試み，情報処理学会論文誌，Vol. 54, No. 12, pp. 2518–2527 (2013).
[10] 高橋真奈茄，小出　洋，近藤秀樹：実践的プログラミング教育の支援環境における活動履歴を用いた活動パターン抽出の試み，情報教育シンポジウム論文集，Vol. 2017, No. 40, pp. 247–254 (2017).
[11] Heinonen, K., Hirvikoski, K., Luukkainen, M. and Vihavainen, A.: Using codebrowser to seek differences between novice programmers, *Proceedings of the 45th ACM technical symposium on Computer science education*, pp. 229–234 (2014).
[12] Toll, D. and Wingkvist, A.: Visualizing Programming Session Timelines, *Proceedings of the 11th International Symposium on Visual Information Communication and Interaction*, pp. 106–107 (2018).
[13] 江木鶴子，竹内　章：プログラミング初心者にトレースを指導するデバッグ支援システムの開発と評価，日本教育工学会論文誌，Vol. 32, No. 4, pp. 369–381 (2009).
[14] 東本崇仁，赤倉貴子：プログラムトレース課題の提案と学習支援システムの開発，電子情報通信学会論文誌 D, Vol. 99, No. 8, pp. 805–808 (2016).