初学者のための漸進的プログラミング学習支援システムの開発

吉田 賢志朗† 松澤 芳昭†
Kenshiro Yoshida† Yoshiaki Matsuzawa†
†青山学院大学　社会情報学部
School of Social Informatics, Aoyama Gakuin University

要旨
初学者のコーディング方法の改善を目指した，漸進的プログラミング学習を支援するシステム「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，1)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こと，2)保存したコードは常に取り出すことが出来ること，の 2 点である．現在，社会情報学部の 1 年生のプログラミング導入授業である，コンピューティング実習に「トリップコード」を導入し，効果の検証を行っている．

1. 研究背景

プログラミング導入教育の現場では，一度に多くのコードを書いてしまったために，多くのエラーが出てしまい，途方に暮れてしまう学生が散見される．そのような学生は，一度に多くのコードを書いてしまうと，どこにエラーがあるのかを特定することが難しいため，途方に暮れてしまうと考えられる．

我々は，一度に多くのコードを書かず，動作チェックをしながら少しずつ構築していくことを，プログラミング導入教育において教育することが重要ではないかと考えている．それにより，エラーが出たときにその問題を特定しやすくなり，エラーを解決しやすくなることで，着実に問題を解き進めることが出来る．

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する．

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[1]は，実際に探索的プログラミングを促進するシステムを開発し，実際にシステムを利用した学生が，ソースコードの保存回数や，プログラムの実行回数が減少することを報告している．

袴田ら[2]は，初学者用のデバッガシステムを開発し，学生に使用させている．対象となった学生に対し，使うことを強制せずとも，7 割以上が利用し，難しい問題ほどシステムを使う学生が増えている．また，アンケート結果から，デバッグ時よりも，自身のプログラムの動作を確認するためにブレークポイントを動かすケースが多いということを，報告している．

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，例を用いて説明する．例として，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，図 1 に示すような 10 個の四角形を描画させる，という問題を考える．

図 1. 10 個の四角形

漸進的プログラミングとは, 完成形に至るまでのステップを作成する, プログラミング手法である. 図 1 の問題の場合, 正答は図 2 のようなソースコードになる. ステップに分けると,

S1. 四角形を描く(3~6 行目)
S2. 次の四角形を描き始める場所まで移動させる(7~11 行)
S3. その動作を 10 回繰り返す(2,12 行目)

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プロセスを, ステップと定義する.

```
1  var t = createTurtle();
2  for (var j = 0; j < 10; j++) {
3    for (var i = 0; i < 4; i++) {
4      t.fd(20);
5      t.rt(90);
6    }
7    t.up();
8    t.rt(90);
9    t.fd(30);
10   t.lt(90);
11   t.down();
12 }
```

図 2. ソースコード(パターン 1)

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.

3.2. 漸進的プログラミングを行っていない場合

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を図 3 に示す. この例では, 図 2 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, どこまで出来ているか分からず, 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.

```
1  var t = createTurtle();
2  for (var j = 0; j < 10; j++) {
3    for (var i = 0; i < 4; i++) {
4      t.fd(20);
5      t.rt(90);
6    }
7    t.up();
8    t.rt(90);
9    t.fd(30);
10   t.down();
11 }
```

図 3. ソースコード(パターン 2)

3.3. 漸進的プログラミングを行っている場合

漸進的プログラミングを行っている学生は, 3.漸進的プログラミングとは, の S1~S3 のように, 小規模なステップを作成する. それにより, 図 3 のようになる前に, 図 4 のソースコードを実行する. その場合, S2 のステップをクリアしていないため, S3 に進むことなく, S2 をクリアするまで編集・実行を続ける.

漸進的プログラミングを行っている場合, S1 はクリアしたが, S2 はクリアしていない, という様に, どこまで出来ているかが分かる. よって, 図 4 の時点で実行結果を確認し, 足りない箇所を判断して修正することが出来る.

```
1  var t = createTurtle();
2  for (var i = 0; i < 4; i++) {
3    t.fd(20);
4    t.rt(90);
5  }
6  t.up();
7  t.rt(90);
8  t.fd(30);
9  t.down();
```

図 4. ソースコード(パターン 3)

4. 提案システム

4.1. システム概要

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. 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, プログラミングのプロセスが可視化され, 漸進的プログラミングの促進をするシステムである. トリップコードの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. 本システムは大まかに以下の 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　手戻り機能：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 は, 学生が解く上で, 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．

機能２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，解き進めた過程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，以前のコードを表示し，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，確認することが出来る．

4.2. インタフェース

開発した「トリップコード」は，JavaScript の学習環境である「Sumatra」に組み込まれている．Sumatra の外観を，図 5 に示し，トリップコードの外観を，図 6 に示す．

Sumatra では，ソースエディタにコードを記述し，実行することで，タートルキャンバスに結果が表示される．実行ボタンを押すと，自動でコードが保存され，トリップコードのプロットが，右に 1 つ追加される．

トリップコードは，折れ線グラフの外観になっており，縦軸が保存時のコードの行数，横軸はバージョン(保存回数)を表している．赤いプロットは，ソースエディタに表示されているコードのバージョンを表している．図 5 の場合，4 回目に保存したコードが．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．「<」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．

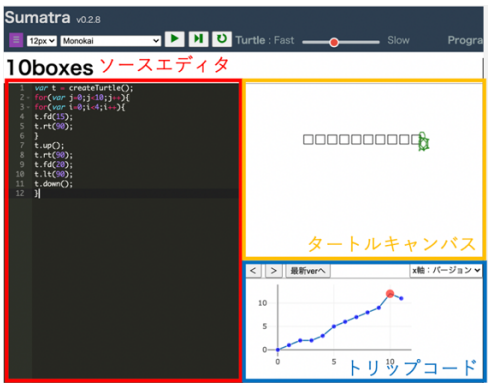

図 5. Sumatra

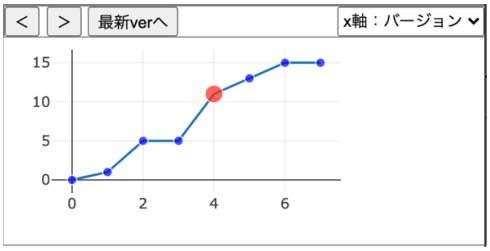

図 6. トリップコード

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. 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，教師が履歴表示機能を使うシナリオについて説明する．

A 先生は，授業で学生の解答をチェックしている．ある時，学生の B 君から，どこが間違えているか分からないという質問が来た．そこで B 君のトリップコードの形を確認したところ，図 7 のようになっていた．2 回目の保存からから 3 回目の保存にかけて，コードの行数が 10 行と，大きく増加していることが分かる．このグラフから，B 君は，小規模なステップを作成しておらず，一度に全てをコーディングしてしまっていると考えられる．トリップコードの形から，一度に大量のコードを書いてしまい，直す箇所が分からなくなってしまったのだという仮説を立て，A 先生は，解き方の指導を行うべきだと判断した．

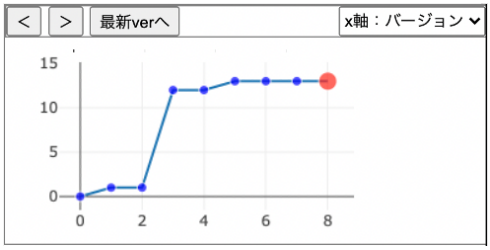

図 7. トリップコード(パターン 1)

漸進的プログラミングを行っている学生のトリップコードの形は図 8 のように，なだらかになっている．小規模なステップに分けている場合，一度に増加するコードの行数は 2~4 行程度であり，図 7 のように，一度に 10 行以上，コードの行数が増加するはずがないからである．

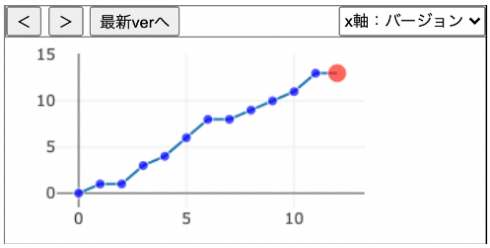

図 8.　トリップコード(パターン 2)

履歴がグラフ形式で可視化されているため，学生が一度に大量のコードを書いていることを，一目で認識することが出来る．このようにして A 先生は，トリップコードの履歴表示機能から，その学生の解き方の仮説を立て，漸進的プログラミングの指導に繋げることが出来た．

4.3.2. 手戻り機能の使用シナリオ

本節では，学生が手戻り機能を使うシナリオについて説明する．

C 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 9 のようになった．C 君は S2 の，次の四角形を描き始める場所まで移動させるステップまではクリアしており，後は 10 回繰り返すだけだと考えていたが，想定外の実行結果が出てしまった．どうやら，S2 のステップをクリアした気でいたが，間違えているらしい．

図 9.　実行結果(パターン 1)

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 10 のようになっている．一見すると，S2 のステップをクリアしているように思える．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 S3 の，動作を 10 回繰り返すステップを行ったところ，正しい実行結果を出力することに成功した．

図 10.　実行結果(パターン 2)

このようにして C 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，C 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．

5. 今後の展開

トリップコードを使って，プログラミング初学者に漸進的プログラミングを行わせることが出来るか実験するため，青山学院大学社会情報学部社会情報学科の 1 年生約 200 人を対象としている．現在 JavaScript 演習授業である，コンピューティング実習の授業で，実験中である．今後は，使用者アンケートや，ログの解析による効果の検証を行う予定である．

参考文献

[1] 槇原 絵里奈, 藤原 賢二, 井垣 宏, 吉田 則裕, 飯田 元(2016)：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
[2] 袴田 大貴, 松澤 芳昭, 太田 剛(2014)：初学者向けデバッガ DENO の利用実態の分析