ビジュアルプログラミングとテキスト記述形式の連携による初学者向けプログラミング学習システムの提案

山梨 裕矢[1,a)]　佐々木 晃[2,b)]

概要：ビジュアルプログラミング言語 (VPL) はプログラミング初学者にとって有用であるが，テキスト記述形式のプログラミングを学べないという欠点がある．それを解決するためにテキストエディタを備えた VPL システムも研究されているが，テキスト表現と VPL との表現の関連性が分かりにくい．そこで，本研究ではブロックによる VPL 表現とテキスト表現の間に両表現を併用して記述できる記述形式を取り入れたシステムを作成した．これにより，初学者は VPL 記述からテキスト記述へ完全に移行する前に，テキスト記述の経験を得ることができ，またテキスト記述でつまずきがあり VPL へ戻りたい場合に VPL での学習に戻りやすくなっている．また，ライブプログラミングを統合し，実行結果をソースコード編集時に即座に実行結果としてみせることで各命令の振る舞いも学べるようになる．両表現で記述している場合ライブプログラミングはコードの誤りをいち早く学習者に知らせるため，テキスト記述で間違えた箇所はまたブロック学習に戻ることができる．一つの画面内にテキスト，VPL，実行結果のすべてを編集動作と結びつけながら表示することによって，各対応関係が学習者に伝わりやすくなった．

キーワード：ビジュアルプログラミング言語，テキストとの相互変換，ブロック型言語，ライブプログラミング

1. はじめに

初学者向けのプログラミング学習環境としてビジュアルプログラミング言語 (VPL) が用いられることがある．ブロック言語に代表される VPL はタイピングを必要としないため子供や初学者にも扱いやすいという利点があるが，主流のプログラミング言語はテキスト記述形式であり，VPL で学習をしてもいずれはテキスト記述形式で学びなおさなければならない [1]．そこで本研究ではブロックによる VPL とテキスト記述形式を連携させ VPL を用いた初期学習からテキスト記述形式のプログラミング学習までを一つのシステム内で実現するプログラミング環境を提案する．具体的には，VPL でプログラムを組むとそれに対応するテキスト表現を出し，反対にテキスト形式で打ち込んだプログラムを VPL として出力することによりテキストと VPL の連携を実現する．テキストから VPL へ変換する時，従来はテキストエディタと VPL エディタが分離され，テキストエディタに打ち込んだプログラムが VPL エディタの方に変換され出力

[1] 法政大学情報科学研究科
[2] 法政大学情報科学部
[a)] yuya.yamanashi.9a@stu.hosei.ac.jp
[b)] asasaki@hosei.ac.jp

されることが多かった．しかし，本システムではVPL エディタに直接テキストを打ち込めるようにし，そこでテキストから VPL への変換を行わせることで 2 種のエディタの距離的ギャップをなくすことによって，VPL からテキスト記述へのシームレスな学習移行を可能にする．また，ソースコードを編集すると同時にプログラムの実行結果を即時に反映させ表示させるライブプログラミングを取り入れることによって静的な表現であるプログラムコードとその動的な表現である実行結果を結びつける．これにより VPL でプログラムを組みながらテキスト記述だけでなくプログラムの挙動も学べるようにする．現段階ではブロック言語によるプロトタイプを作成したためこれを報告する．

2. 研究背景

初学者向けのプログラミング言語にビジュアルプログラミング言語がある．これはプログラムの各命令を視覚的なオブジェクトで表し操作できるようにしたものである．多くは視覚的なオブジェクトにブロックを採用し，ブロックの組み合わせでプログラムを組めるようにしている．従来のテキスト記述形式のプログラミングでは文法や構文を学ばなければプログラミングができなかったものが，ブロック型の VPL では与えられたブロックを組み合わせることでプログラミングが可能であるため，子供や初学者でもすぐにプログラミングを始められる利点がある．

その一方で，C や Java といった今日主流なプログラミング言語はほとんどすべてがテキスト記述形式のプログラミング言語である．したがって，VPL だけでプログラミングの方法を学んだ場合，いずれテキスト記述の仕方を学ばなければならなくなる．この問題点を解決するために，VPL だけでなく同じシステム内にプログラムのテキスト記述形式も持たせる必要があると考えた．しかし，それだけでは VPL での視覚的表現とテキスト表現の隔たりが大きくプログラミング初学者が視覚的表現とテキスト表現の対応関係をつかむことができず，VPL からテキスト記述形式への学習移行が行われないのではないかとも考えた． そこで，VPL 記述形式とテキスト記述形式の間に中間状態を取り入れることによりテキスト記述形式へのよりスムーズな学習移行をはかる．

3. 先行研究

3.1 Hinoki[1]

ブロック型言語からテキスト記述形式へのシームレスな学習移行を図ったプログラミング環境であり，オブジェクト指向構文の Java とブロック型言語の相互変換に対応している．VPL とテキスト記述の両方を備えており相互変換が可能である点は本システムと同じだが，相互変換がリアルタイムに行われないことと両表現が同一画面に表示されずブロックとテキストの対応関係が見づらい点が欠点としてあげられる．

3.2 Tiled Gracegrace

Tiled Grace はブロック型言語とテキスト記述形式の両方を備えたプログラミング環境である．このシステムも VPL 記述形式からテキスト記述形式への移行を目的に作られており，テキストエディタと VPL エディタをアニメーションで変換させることによりテキストとブロックの対応づけを示している．しかし，両表現が同一画面に出てこないことと，すべてのブロックが一度にテキストに変わってしまうので対応関係がわかりづらいという欠点がある．

4. 提案手法

ブロック型言語によるプログラミングとテキスト記述形式の両方でプログラミングができ，二つの表現間に連携を持ったシステムを提案する．初学者は，まず文法を覚えたりタイピングを必要としないブロック型の VPL で慣れ，その後，徐々にテキスト形式も扱いながら，最終的にテキストのみでのプログラミングへ移行できるようにテキスト記述形式でのプログラミング環境も備える．これを実現するために具体的には，VPL 表現とテキスト表現の間に両表現を併用して記述できる記述形式を取り入れる．テキスト記述の経験がないまま VPL からテキスト記述へ移行するのは初学者

にとって困難なため，この両表現の併用でテキスト記述の経験を積み，完全なテキスト記述形式への移行の手助けとする．

4.1 VPL とテキスト記述の双方向変換

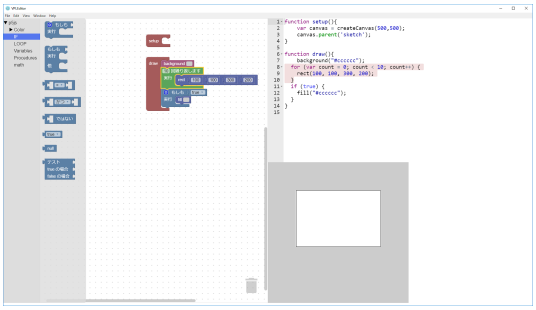

図 1 システム全体図

図 1 の左側はブロック型 VPL でプログラミングができるようになっており，プログラムの命令列をブロックで表している．左にあるブロックリストからブロックを取り出してきてワークスペース上で組み合わせていくと，画面右のテキストエディタの方に対応するコードがリアルタイムに表示される．また直前に操作したブロックに対応するテキスト表現を画面右のエディタ内のように赤くハイライトさせることで各ブロックに対応するテキストでの書き方を学習者に分かりやすいようにした．画面右下には現在記述してるプログラムの実行結果が表示される．

また，本システムではテキストからブロックへの変換も行われる．編集したいソースコードファイルを読み込み時にそれに対応したブロックを出力する．また，テキストエディタに命令列を入力した場合にもブロックがリアルタイムに出力される．構文エラーがある場合はブロックに変換されない．

4.2 4.2 VPL 記述形式とテキスト記述形式の併用とその利点

VPL での記述の仕方とテキストでの記述の仕方の差異は大きく，テキストと視覚的表現の対応関係がわかったとしても，初学者が VPL 記述形式からテキストエディタ上でのテキスト記述形式へ移行するのは困難である．そこで，VPL エディタ上でもテキスト記述を受け付けるようにし，一つの命令ごとにもブロックに変換するようにした．VPL エディタ上の任意のスペースでダブルクリックをするとテキストフィールドを表示し，そこにプログラムの命令を打ち込むと対応するブロックが出力される．また，テキストフィールド出現時に結合したい先のブロックを指定しておくと変換後に結合される．図 2 にテキストフィールドの表示からブロックの結合までの流れを示す．

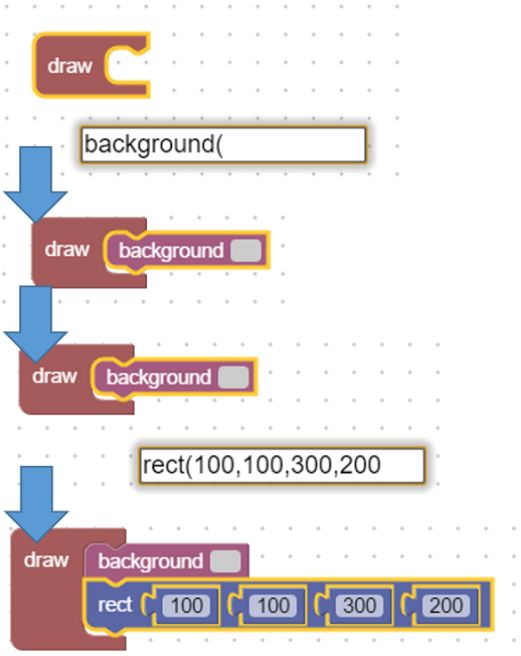

図 2 ブロックエディタ内でのテキスト入力

このように VPL とテキスト表現の両方を用いて記述するメリットは二つ考えられる．まず，ブロック型 VPL 特有のプログラミングに時間がかかる問題をなくすことができることがあげられる．ブロック型 VPL でのプログラミングでは，各命令をブロックリストから選択し，取り出し，組み合わせるという操作が必要になるためテキスト記述より時間がかかる．たとえば，計算式で「2+3*4」と入力したい場合，数字ブロック 3 個と演算子ブロック 2 個を取り出してきて組み合わせる必要が

ある．また，ブロックリストのブロックが多い場合，目的のブロックを探すのに時間がかかる．この問題については，テキストと併用することにより，ブロックを探す手間や組み立てる時間を減らすことで解決できる．さらに，テキスト記述を用いらせることによってテキスト記述のみに移行する前段階からテキスト経験を得ることができる利点もある． 次に，テキスト記述形式から VPL 記述形式に戻りやすい点が挙げられる．まだテキスト記述形式へ移行するには早い段階でテキスト記述形式に挑戦してしまった場合，構文エラーなどをおこし VPL 記述形式に戻って学習したいと考える場合が出てくる．その時に，テキスト表現と視覚的表現が完全に独立していると，エラーが起こったプログラムはすべてブロックに変換されない，もしくはエラーが起こった以降の命令が変換されず VPL 記述形式に戻ることが困難になってしまう．しかし，テキスト記述形式と VPL 記述形式を併用するとインクリメンタルにブロックが足されていくので，途中でつまずいてもいま入力された箇所だけが変換されないだけなので間違った箇所がわかりやすく，すぐに VPL 記述形式に戻ることができる．

以上二つ点の利点によってテキスト記述の経験を段階的にへることによって，VPL 記述形式からテキスト記述形式への学習移行をはかる．

4.3 ライブプログラミングの統合

プログラムの振る舞いに関する理解支援手法にライブプログラミングがある [3]．厳密な定義がされていないが一般的に，プログラムの編集時に即座にプログラムの実行結果や実行状態をユーザに提示するものがライブプログラミングと呼ばれており，本研究ではこれをライブプログラミングとしている．テキスト表現や視覚的表現はプログラムを静的にしかユーザに見せることができなかったが，ライブプログラミングによりコードを動的に見せることができる．

テキスト記述でつまずいたときに VPL 記述に戻ることが容易であることは前節で述べたが，ライブプログラミングはこの「つまずき」を発見するのに役立つ．つまずきを早期発見し VPL に戻るきっかけを与えることでテキスト記述と VPL 記述の連携をより高める．ブロックとテキストの両方でプログラムを記述する場合，前述の通りインクリメンタルに命令を追加していき，合わせて修正することもできた．これにライブプログラミングを取り入れることで命令ごとの実行状態もわかり，もしエラーが起きたり期待しない結果になったりしてもすぐにわかるのでプログラムが間違っているのに気づくことができる．プログラムの間違いを修正する際，テキスト記述形式の場合では変更したい命令の書き方を参照したりテキストの打ち直しでタイピングをしなければならないが，ブロック型 VPL の場合ブロック欄から適当にブロックを取り出したりブロックの付け替えだけでそれが実現できるためテキスト記述形式より修正しやすい．

5. 実装

システム全体は JavaScript で実装しており，Electron を用いてアプリケーション化している．VPL 部は Google が提供している JavaScript ライブラリである Blockly を使用している．Blockly で使われているブロックには各ブロックにそれに対応するテキスト型のコード断片を出力するコード生成プログラムが設定されており，ブロックの構成に応じたテキスト表現が出力される．

テキストからブロックへの変換は，テキストエディタに入力されたソースコードに対応する抽象構文木を用いて行う．ソースコードから抽象構文木への変換は JavaScript の構文解析器 esprima で行い，生成された抽象構文木に対し estraverse を用いて各ノードに対応するブロックを出力する．

VPL エディタ上で入力されたテキスト記述形式のプログラムをブロックに変換する際にも同様の操作を行っている．もし VPL エディタ上でテキストからブロックへの変換を行う際には，既に配置されているブロック列に直接挿入できる．選択されているブロックがある場合，次のブロックを受け付ける場合や下の階層にブロックを受け付ける場合はその下に接続させる．

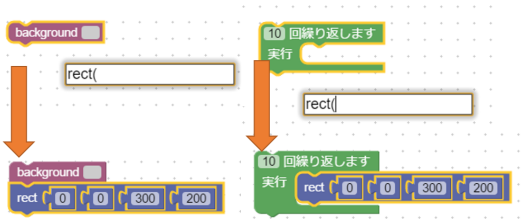

図 3　ブロックの形による接続場所の違い

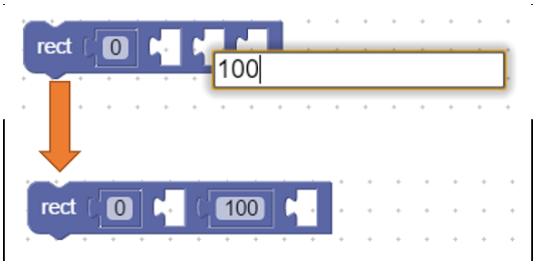

図 4　ブロックの内部挿入

この時接続の優先順位は下の階層の方を優先させているが使用頻度がそちらの方が高いと考えたからである．たとえば，for 文の次に命令を接続する場合より for 文の中に命令を入れることが多いと考えられる (図 3)．ブロックの内部にブロックを組み合わせることのできる場合はテキストフィールドから一番近い入力場所を検索し，そこに入れこませるようにしている (図 4)．

本実装では，JavaScript(p5.js) の学習を行うことができる．p5.js は Processing 言語を JavaScript でも書けるようにした JavaScript 用ライブラリである．Processing 言語は視覚的なフィードバックを得るプログラムを簡単に書けるようになっておりプログラミング初学者向けの言語と言われておりライブプログラミングとも相性がいいため本システムではこれを採用した．p5.js は JavaScript のライブラリであるため，p5.js で定義されている言語だけでなく JavaScript で使える構文はすべて使えるようになっている．

6.　考察

VPL からテキスト記述形式への学習移行をスムーズに行うために，本研究では VPL エディタ内でもテキスト記述を行えるようにした．このように VPL とテキスト記述の中間段階を設けることにより，テキスト記述に完全に移行する前段階でテキスト記述の経験を得ることができるため初学者のよりスムーズな学習移行を促進する．テキストからブロックへの変換を 1 命令ごとに行うことで，テキストから正しいブロックに変換されていることを学習者は確認することができ，構文エラーが起こった場合にはそのブロックへの変換が行われないのみでほかのプログラムのブロック変換へは影響がでない利点がある．また，構文エラーだけでなく，ライブプログラミングと組み合わせることによって予期せぬ結果が起こった場合にも学習者はすぐにそれを確認することができる．テキスト記述での修正が困難な場合にはテキストよりも修正の容易なブロック記述に戻ることができ，テキスト記述学習でのつまずきを大きくさせない効果がある．

ほかにも同一画面内にテキストエディタとブロック言語のエディタを表示させ，リアルタイムに相互変換を行ったりブロックから変換されるテキストをハイライト表示させることにより先行研究のシステムよりブロックに対応するテキストが分かりやすくなっていると考える．さらに，ライブプログラミングを行うことでブロックに対応するテキストだけでなくその挙動までも確認することができて，プログラムの書き方だけでなく振る舞いも学ぶことができる．

発展性としては以下のようなものが考えられる．まず，ブロックとテキストの対応付けの強化があげられる．本システムでは選択しているブロックに対応しているテキストがハイライト表示されていたが，その逆は出来なかった．テキストエディタで選択している箇所に対応するブロックをハイライトすることができれば，テキストとブロックの対応のわかりやすさが強まる．また，現在はテキスト記述が完了した後にそれに対応するブロックが表示されるようになっているが，テキスト記述中から挿入されると予想される位置に候補となるブロックを表示することが考えられる．テキスト記述時から挿入される予定のブロックの場所を示すことによりテキストでの誤った操作防止を図る

と同時にテキストとブロックの対応関係も強まる．

ライブプログラミングはプログラムの動的側面を見せるのに役立ったが，これにプログラム編集前と後の実行画面を見せることでより自分が編集したプログラムの変更を確認できるようになるのではないかと考える．

現在想定される問題点を挙げる． テキストプログラムに加え VPL と実行結果のすべてを一つの画面内に表示させるため，それを表示できるだけの広いスペースが必要となる．ブロックとテキストの対応関係を学習者に示しやすいように本システムでは両表現をタブなどに分けず同一画面内に同時に表示させている．また，ソースコードの動的な側面の理解のためにライブプログラミングを取り入れ，エディタと実行結果も同一画面に表示させている．このため，大きなプログラムになるとそれぞれの表現の対応関係が分かりづらくなってしまう．解決策として，三種の表現の中で一番場所をとる視覚的表現の見せ方を工夫する方法が考えられる．たとえば，タイルを小さくし必要になった場合や注目したい場合のみ拡大し強調させる工夫などがあげられる．この必要になったタイルのみ拡大させる方法は必要なもの，強調させたいもののみをハイライトすることにより，実行時表現やテキスト表現とのつながりをより良くみせることができるのではないかということも考えられる．

7. まとめ

VPL 記述とテキスト記述の両方を備えたプログラミング環境を作成した．プログラミング学習者が VPL での記述からテキスト記述へ移行できるようにするため，VPL での記述とテキストでの記述を一つのエディタ内で使えるようにした．VPL 記述から直接テキスト記述へ移行するのではなく，この中間状態を経ることによりテキスト記述に慣れることで完全なテキスト記述に移行できるようにした．これにライブプログラミングを取り入れることは，プログラムの動的な側面を見せるだけでなく，早い段階でエラーを発見し学習者に報告することで修正の容易なブロック言語に戻ることができテキスト記述でエラーを起こしどこを修正したらよいかわからなくなることを防ぐ．

その一方でテキストエディタや VPL エディタ，実行結果を一つの画面内におさめたために，各表現のつながりが分かりやすくなる反面，大きなスペースを必要とする問題もあった．またライブプログラミングにより命令数の多いプログラムだとシステム全体が重くなる問題もあった．初学者向けから一般向けのシステムにする場合これらの問題を解決する必要がある．

発展性としては，テキストエディタで選択したコードに対応するブロックをハイライト表示させたり，テキスト記述でコーディングしている最中にブロックの挿入予定位置や挿入されるだろうブロックの候補を表示することによってテキストとブロックの対応を強め完全テキスト記述形式での学習をより分かりやすくしていくことが考えられる．

参考文献

[1] 大畑貴史, et al. ”BlockEditor Hinoki: ビジュアル-Java 相互変換技術を利用したオブジェクト指向プログラミング教育の提案.” 情報教育シンポジウム 2014 論文集 2014.2 (2014): 35-42.

[2] HOMER, Michael; NOBLE, James. Combining tiled and textual views of code. In: Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on. IEEE, 2014. p. 1-10.

[3] 大畑貴史, et al. ”BlockEditor Hinoki: ビジュアル-Java 相互変換技術を利用したオブジェクト指向プログラミング教育の提案.” 情報教育シンポジウム 2014 論文集 2014.2 (2014): 35-42.