目次

第１章
序論

　現代において、情報技術は生活に欠かせない技術となっている。これまでは人力や原始的な構造で動いていた様々なものが情報技術により制御され、駆動している。
/* 実例 */
　その情報技術の発展にエンジニアの存在は欠かせず、専門的な技術を持ったプログラマたちが今の社会の基盤を創り上げたと言って問題はないだろう。それほど現代と情報技術は深く関わっている。しかし時代は進み、必要な人材も変化しているのではと私は考えた。まだ情報技術と深く関わっていない分野、言うなれば新たな市場を発見し、開拓できるような人材がこれからの発展に有用だと言えるのではないか。/* 具体例。Suicaとか？ */無論、基礎的な情報技術の革新は限界には未だ遠く、通信速度、処理速度、小型化、軽量化など、より高性能な技術に向けて開発が進んでいる。それは近年のPCやポータブルデバイスの進化からも見て取れる。しかし、これらの技術者はコアであるがゆえに他の分野との関わりが薄い。通信と電波等、近い分野ならその限りではないが、新たな分野の開拓に進むのは難しいだろう。またアイデアがあったとしても、それを実現できる環境ではない。
　情報技術というものは学問の一つでありながら一つのツールであり、様々な分野に応用できる多くの可能性を持っている。第一次産業である農業を例にとってみる。まず生産段階で、ITを利用したセンサや機械による農作物の管理が可能である。降雨量や湿度から栄養、水分量を調節することができる。流通に関しても、現在Web上のサービスにより多くの作物が売買されている。ローカルな特産品と現地に行かずに情報を得られるインターネットは相性がよい。このように多くの分野にITを組み込むためには、様々な分野で仕事をする人間が、基礎的な情報技術、もしくは情報技術に対する一定の理解を持っていることが必要であるのではないか。
　そこで、本研究では「プログラミングをより理解しやすくするにはどうするか」を主題として、プログラミングに対する意識調査、そしてビジュアルプログラミングツール「Visual-Poyogramming（以下VP）」の実装を行った。調査はプログラミング初学者が理解しづらい場面、箇所はどこか、そして学習意欲を削ぐ要因は何かについてのアンケートを実施した。またツールの作成に関しては、プログラムの構造をビジュアル化し、複雑なコーディングを可能にするよりはむしろアルゴリズム構築の基礎を学んで

もらうことを目的とした。また、「可視化された環境でのプログラミング」が理解にどのような影響を及ぼすかに対する調査も兼ねている。

第２章ではプログラミングに対する理解と、モチベーションを重視したアンケートによる考察を行う。また、先行研究や実際に稼働している初学者向けのサービスと合わせ、どのような形式、特徴を持ったものが初学者に優しく理解しやすいのかも考慮する。

第３章では実装したビジュアルプログラミングツール「VP」の解説を行う。構成や実装した機能、また利用者によるフィードバックを述べる。

第４章では全体を総括して今後の課題を述べる。本ツール「VP」はプロトタイプであり、今後実装するツールに向けどのような機能、デザインが望ましいのかをこれからの情報教育の展望と合わせて考察する。

第２章
プログラミングに対する意識調査

2.1　概要

この章ではプログラミング初学者が実際にプログラミングをどう捉えているかについて調査し、問題解決に何が必要かを考える。調査はアンケートを実施し、プログラミングをどのレベルまで理解しているか、またプログラミングに対しどのような感情を感じているか、作業が停滞する原因は何かを調べた。

2.2　調査結果

まず、プログラミング技術に関するアンケートについて考える。項目は以下のとおりである。これらの質問について、自身があるかを回答してもらった。回答はできる、おそらくできる、不安がある、できないの４段階とした。またこの回答を数値化し、その平均値を合わせて記載しておく。なおこのアンケートの対象はSFCにおけるプログラミングの初学者である。プログラミングの経験が半年以下の学生が多くを占めており、彼らの学んでいる言語はJavascriptである。

	回答
1,「変数について、人に説明できるか」	2.5
2,「代入について、人に説明できるか」	2.75
3,「条件分岐（if文）について、人に説明できるか」	3.375
4,「while文について、人に説明できるか」	2.25
5,「for文について、人に説明できるか」	1.75
6,「プログラミングの構造は簡単だと思うか」	2.0
7,「プログラミングは好きか」	1.875

表１

今回、質問の形式を「理解しているか」ではなく「人に説明できるか」とした。理由としては「できる」と「おそらくできる」の差別化である。ある程度自身の技術として持っていないと人に説明できるとは選びづらく、「理解している人」と「なんとなく理解している人」を区別することができると考えた。

結果としては変数、代入に関しての回答が条件分岐に関する回答より自信がないということになった。これは変数、代入に不安がある人が多いのではなく、if文なら説明できるという回答の人が多かったためである。条件文に関する処理は他の部分に比べわかりやすいと感じるようである。それと反してwhile、for文の理解は浅い生徒が多かった。条件文はプログラムの上部から流れていく動作に反する動きをしないため理解しやすいが、繰り返しの処理が見えないのが分かりにくさの原因となっているのだろう。

また構造は難しいかという質問に対し、やはり難しいと感じる生徒が多かった。上記の結果と合わせると、やはり繰り返しの処理や複数のif文の組み合わせが難しいと感じられるようである。それぞれの構文への理解と、その組み合わせによる処理が可視化できれば解決に近づけると考えられる。

次にプログラミングに対する意識についてのアンケートについて考える。こちらは具体的な事例を回答として受け取った。

1, 「プログラミングにおいて複雑だと感じるのはどのような箇所、場合か」
2, 「プログラミングにおいて、何度も躓くのはどのような箇所、場合か」
3, 「プログラミングにおいて、
モチベーションが下がると感じるのはどのような箇所、場合か」

表２

以上３点について質問を実施した。これにより初学者がプログラミングに複雑さを感じている点が何かを考える。まず多かった意見が、「中括弧による構造構築の難しさ」だった。Javascriptはブロックの構築に中括弧（”{“ ,”}” ）を用いており、中括弧で括ることでその内部が一つのブロックとなっていることを表現している。プログラマは字下げスタイル（Indent Style）を使用することでブロックの構造を視覚的に確認できる形で記述する場合が多い。だがこれはあくまでユーザー側の工夫であり、プログラムとは無関係のものである。初学者はこの技術を知らず、字下げが不十分なコードを

書いてしまう。結果として構造が見辛くなり、ブロックの構造が崩壊する。具体例としは「中括弧のどちらかが多い、少ない」といったものとしてエラーが現れる。

次に「スペルミス」についての意見があった。変数、関数を呼び出すときにスペルミスが発生し原因が別にあると感じる、もしくは原因が全く分からずに作業が停滞する。これの理由として、エラーメッセージの意味がわからないという意見が多かった。コーディングに慣れていない初学者は多くの場所でミスをする。小規模でのテストという手法をとらず、複数のエラーの要因をまとめてテストすることが多く、当然一度のデバッグではエラーは除去しきれない。結果、「いくらやってもエラーが消えない。プログラミングのデバッグの作業は大変で面倒。」といった感情を抱いてしまう。初学者にとってエラー出力やコンソール画面は身近なものではなく、今回被験者が利用していたHTMLとJavascriptでは、エラーの発生したWebページは何も表示されない、もしくは動作しない静的なページとなってしまう。「成果が現れる、実感できるまでにデバッグ作業が多く、時間がかかる」のがモチベーションの低下と密接に関係があるようだ。

2.3　プログラミングに対するモチベーション

上記したアンケートの結果から、初学者のプログラミングに対する意識がどのようなものか、またそのモチベーションに関わる要因が何かを考察する。まず、今回の被験者はSFCにおけるプログラミング入門のための授業であり、ほぼ全生徒が履修する授業の履修者である。よって、プログラミングに興味があるかないかに関わらず履修している。学習意欲の高い人ではある程度までの障害によるモチベーションの低下がわかりにくいため、今回はモチベーションの少ない初学者を対象とした。

簡潔な結果としては、「出どころのわからないエラーによる長時間、複数回の作業の停滞」が最も学習意欲の低下に関わっていることが推測される。構文として誤っている箇所がピンポイントでわからないこと、一部の間違いでそれに関わる大部分のコードが動作しなくなること、それに伴う作業の延長が原因と思われる。プログラミングの経験者であればエラーメッセージから予想がつくものも、初学者はそのエラーの原因、コードの箇所がわからない。経験者の補助を受けずに独学でプログラミングを学ぶことは学習意欲の少ない学生からすれば苦難の多い作業となる。

/* そもそもモチベーションの低下とは（時間があれば補足する） */

第３章
ビジュアルプログラミングツール「Visual-Poyogrammingについて」

3.1　概要

本ツール「Visual-Poyogramming」（以下VP）は、マウス操作によるプログラミングを可能にし、初学者にアルゴリズム構築に基礎を学んでもらうことを目的として作成した。第２章において論述した「モチベーションを下げる要因」を取り除いたUser-Interfaceを特徴としている。なお実装に使用した言語はJavascriptであり、ツール内で作成されるコードも同様である。

一つはブロックの形成に関するものであり、中括弧の数の不一致のないプログラミング構成が可能である。これはブロックを形成する文（while文, if文）に他の文を挿入、またブロック外へ他の文を出力することで実現されている。

一つは変数リストの実装である。これにより、利用者は一度宣言した変数をドラッグ＆ドロップを用いて使用することができる。キーボードで変数名を入力することなく代入文や条件文の記述が可能であるため、スペルミスが起こらない仕様となっている。

3.2　使用方法

実装した機能と使用方法の説明を行う。

3.2.1 メインウィンドウ

全体の画面（図１）から、大まかな機能の説明をする。詳しい機能や画面遷移は後述する。まず、①が書いたコードが現れる領域である。②が変数、定数を扱う領域であり、このエリアからドラッグすることで入力が可能となる。③がドラッグされた値を

受け取る領域であり、代入文や条件文を作成することができる。④の領域では、①に制御文を代入することができる。

3.2.2 コードエリア

図１の①にあたる部分である。②、③により作成された代入文や④により挿入される制御文をブロック構造で可視化する。図２はVPにより実際に書かれたFizzBuzz問題の一般的な解答、図３はその解答を実際のJavascriptで書いたコードである。入れ子構造をブロックによる内包と色分けによって表現している。

また行番号が書いて有る箇所をクリックすることでその行を選択することができる。他の選択状態でif文、while文をマウスオーバーしている時、図４のようにExchange、Insertの二つのボタンが表示される。行が選択された状態で他の行をクリック、もしくはExchangeのボタンをクリックすることで行の入れ替えが可能である。なお、ブロックを構成する文とそのブロック内の行は入れ替えができないため、アラートが発生する。もう一つのInsertのボタンを押すことで、その文がもつブロック内に選択された行を挿入することができる。ブロック内の行が選択されている場合、Insertのボタンは Exportボタンとなり、その行をブロック内から取り出すことができる。

3.2.3 変数・定数エリア

図１における②にあたる部分である。「Variables」には変数が、「Other」には定数が挿入できるボタンが設置されている。（図５、図６を参照）これらのボタンから図１における③、アサインエリアへドラッグ＆ドロップすることで挿入が可能である。変数は図４における追加ボタンを押すことで変数名の入力画面へと移り、同じ変数名の変数が宣言されていなければ新しい変数として図５に描画される。またOtherの Number、Stringについては、アサインエリアへドロップされた時にそれぞれ実数、文字列の入力画面へと移る。その画面で正しく入力されると挿入が行われる。

3.2.4 アサインエリア

図１における③の部分である。②の変数定数エリアから値をドラッグして、このアサインエリアで代入文、条件文を作成することができる。灰色の「Null」と記述されているボックス、またすでに代入されている青色のボックスへドロップすることで、ドラッグ元の値を代入することができる。この操作を使い文を構築する。まず代入文を作成するAssignエリア（図７）について解説する。

図中にTargetとあるが、これが代入される変数である。通常のコードではただ左辺として扱われるが、初学者にはそもそも代入という操作の理解が不十分なことがある。よって、代入先と代入される内容を分割することで構文への理解を深める狙いからこのようなデザインとした。Targetの下、灰色の枠線で囲まれた部分があるが、これが代入される内容を記述するエリアである。変数リストと同じく「＋」と書かれた追加のボタンがあるが、このボタンがクリックされることにより演算子と新たなNullボックスのセットが追加される。演算子は「＋、ー、×、÷、％」である。

Target右側の三つのボタンについて説明する。最も右側のゴミ箱のアイコンがある。利用者はこのアイコンへドラッグすることで変数、定数の削除が行える。削除された変数のボックス内にはNullボックスが設置される。またその左側、リロードのアイコンがある。このアイコンではフィールドのリセットを行える。Targetの内容もゴミ箱へドラッグすることで削除できるが、まとめて消去したい時にこのボタンを使うことができる。またその左側、折れた矢印のアイコンがある。これは挿入ボタンであり、これを使うことで実際に代入を行う。なおエリア内に未定義のボックスがあるとアラートが発生する。プログラマはしばしばnull値を代入するが、初学者向けではないと判断しこの仕様とした。

次にConditionエリア（図８）について説明する。ここでは条件文を挿入することができる。Assignエリアと同様にTargetが存在するが、こちらはドラッグではなく実際に代入する条件文をクリックすることで図のように表示される。図７の例は１行目のif文という意味となる。またこのエリアにおける追加ボタンだが、これは条件の追加が可能である。今回は実装の都合により、最大二項による条件設定となっている。図の通り、and（&&）とor（||）の論理演算子を使用することができる。

最後に④のその他のエリアについて説明する。まずNewのページにあるボタン、New IfとNew Whileのボタンだが、これにより初期状態のif文、while文をコードエリアに挿入することができる。またOtherのページにはMake Codeのボタンがある。このボタンは新規ウィンドウを作成し、コードエリアにVPにより描画されているコードをJavascriptの形式で出力する。

3.3　フィードバック

今回実装したプログラムを初学者に見て、使ってもらい、そのフィードバックとして今後に役立つものがあった。一つは機能の欠落についてである。この度の実装の仕様により、Javascriptの基本構文で実装されていない部分がある。具体例としてはfor文、複数の条件による条件文、null値やundefined値の挿入、評価、関数の定義、利用等である。これより高度なものは初学者向けではないとして意図的に省いたものだが、これらの機能は実装するために十分に時間を割けなかった。

一つは繰り返しの可視化についてである。アンケートによる調査でも現れていたが、「条件分岐はわかるが繰り返しの処理が複雑でわからない」という学生が多かった。テキスト上ではただの文字列であり、その中で戻る、また他の行へジャンプするという処理が慣れないようである。コーディングの可視化にとどまらず、生成されたコードに対するビジュアル化にも着手してほしいという意見が多くあった。本ツールの目的として、「可視化された環境において作業することでアルゴリズム構築の基礎を学ぶ」というものがあったが、コーディングだけでは不十分であり、制作物がアルゴリズムとしてどのように動いているのかを小さいスケールで視覚的に理解することが重要であるとわかった。問題に対する答え合わせを行うことで、学生の自信と理解を生むのだろう。人間の理解のメカニズムとの関連付けも合わせて、今後の課題としたい。

また、学生へ「どのようなプログラミングツール、エディタ、システムがあったらプログラミングがすきになるか」という質問をした。意見の一つは、「日本語化してほしい」というものだった。あまりなじみのないアルファベットが理解できない法則で並んでいることが学生をプログラミングから遠ざけているようだ。これについてはまず構造を理解することが重要だと考えた。不規則に並んでいるように見えるためモチベーションが下がっているとすれば、まずその構造、プログラミング言語がどのような成り立ちをしているのかを理解することでこの問題は解決できる。

一つはコードに対する可視化である。上述したように、完成物に対し理解が追いつかないと「よく理解できないが動いている」という状況になる。このような学習を重ねることで、学生がその場しのぎのコピーアンドペーストによるコーディングを繰り返すことが予想できる。そのため、基礎的な学習において「学習意欲の減退が少ない」、かつ「理解しやすい」方法が必要である。

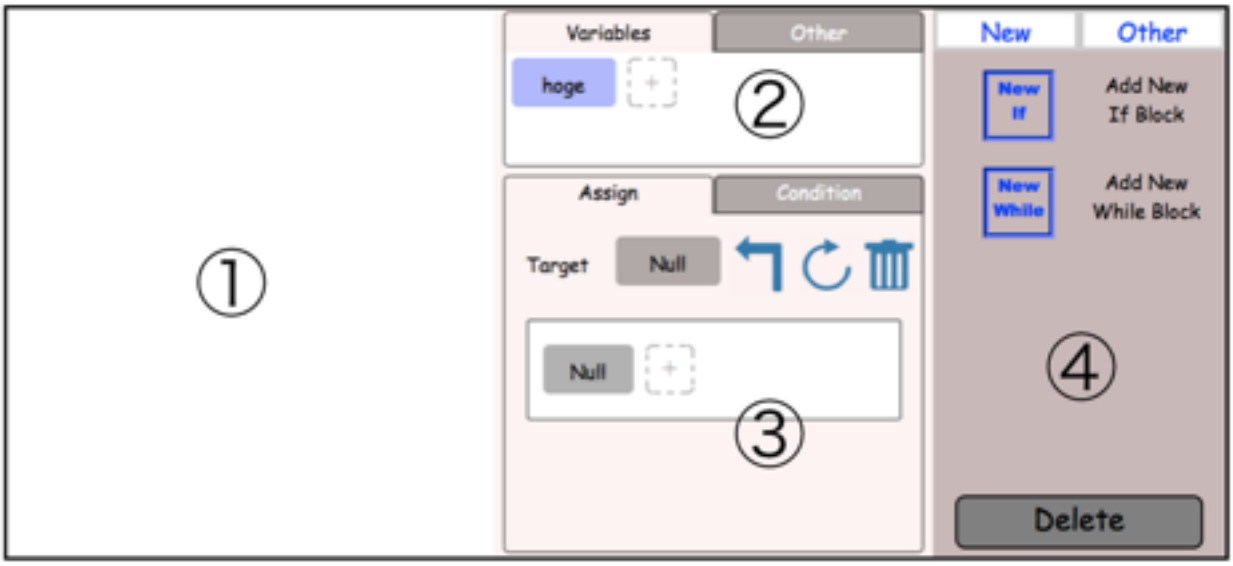

図1

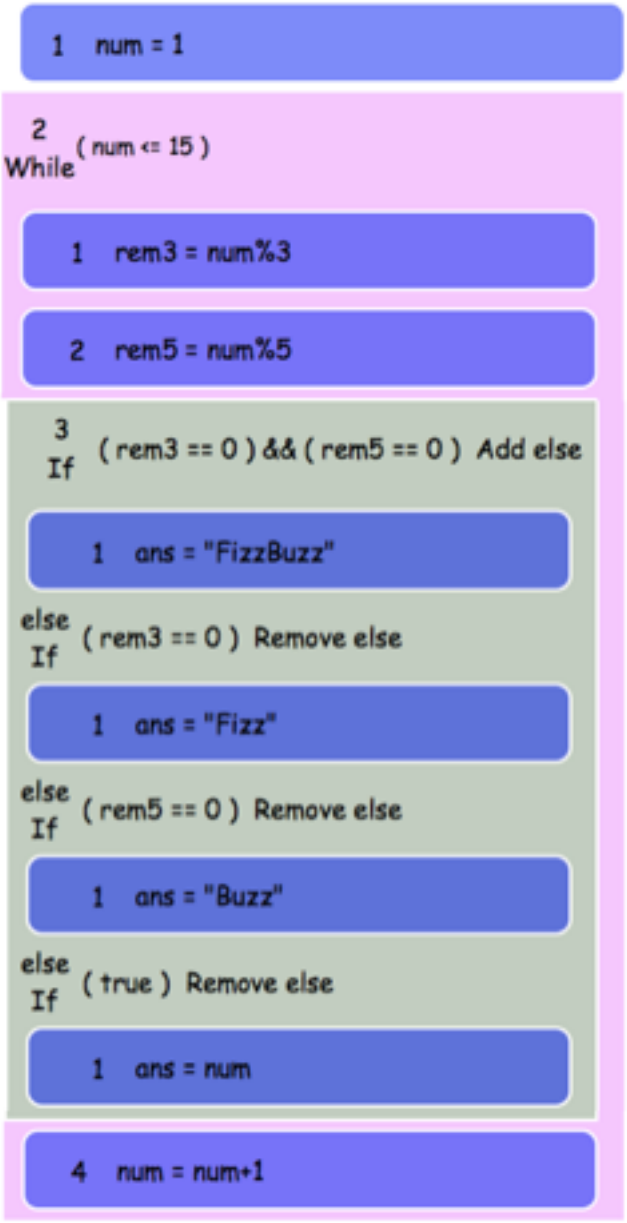

図2

```
num = 1;
while (num<=15) {
   rem3 = num % 3;
   rem5 = num % 5;

   if((rem3==0) && (rem5==0)){
      ans = "FizzBuzz";
   }else if(rem3==0){
      ans = "Fizz";
   }else if(rem5==0){
      ans = "Buzz";
   }else if(true){
      ans = num;
   }

   num = num + 1;
}
```

図3

図4


Variables
Other
hoge

図5


Variables
Other
Number
String

図6

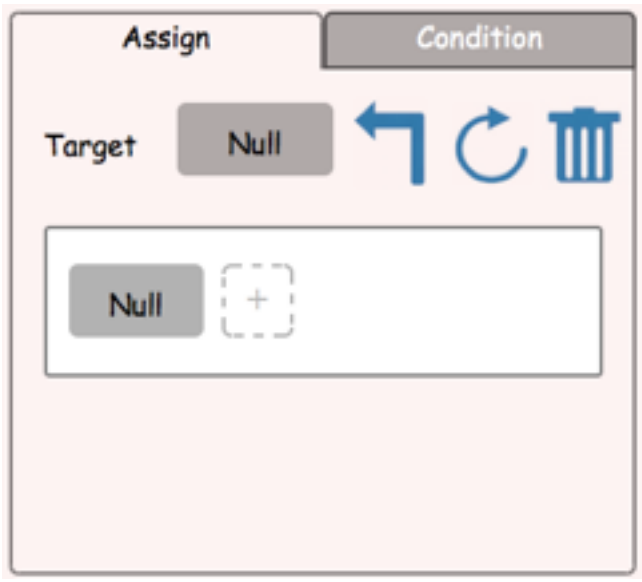
Assign
Condition
Target
Null
Null

図7

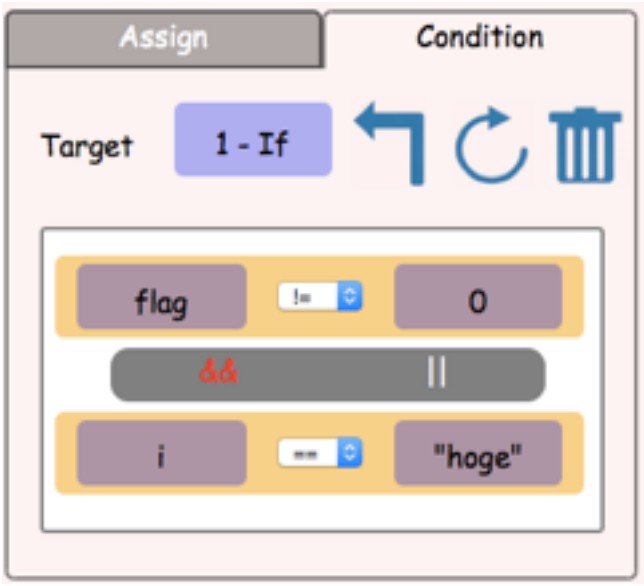
Assign
Condition
Target
1 - If
flag
!=
0
&&
||
i
==
"hoge"

図8

第４章
今後の課題・考察

/* 今回の実装からの改善点 */