プログラミングに苦手意識を抱く学生を対象とした プログラミング授業の提案

慶應義塾大学環境情報学部 4 年
二宮　温子
(70057468,t00755an)

概要

慶應義塾大学湘南藤沢キャンパス(以下、SFC と略す)では、プログラミングの授業の履修がほぼ必修となっている。しかし、授業履修後に自力でプログラムを書けるまでになる学生の割合は極めて低いといえる。授業履修後プログラムを書けるようになる学生と書けない学生の違いは何であろうか。

その理由の一つとして、‘ プログラミング ’ をパソコン・ツールの一つとして捉え、履修前に既に苦手意識を抱いてしまっていることが挙げられる。

しかし、私は全ての学生にプログラムを書けるようになってもらいたい。なぜなら、プログラミングというのは、計画・実装・評価・再構築のプロセスを指すのであり、そのプロセスは、コンピューターの世界のみに限らず、‘ ものをつくる ’ ということ全てに共通しているからだ。

そこで、プログラミングに苦手意識を抱いていた私自身の経験から、学生がプログラミングを苦手とする原因を分析し、それに対してどのように対処すればよいのか、具体的な方法を提示する。そして、プログラミングに対して苦手意識を抱いている学生も楽しめるようなプログラミングの授業を提案していく。

研究会での様々な経験を通じて、私自身の、プログラミングに対して苦手意識を抱いていた原因がよりリアルに、明確となっていった。そこで、この論文はプログラミングに対して苦手意識を抱いている学生にぜひ読んでいただきたい。プログラミングの考え方というのがいかに汎用的で重要なものであるか、そしてその考え方を習得する上でなぜプログラミングが最良の題材であるのかを少しでも理解していただければとても幸いである。

目次

1. 問題意識

1.1.研究会を履修する以前の自分について

私自身プログラミングに対して強烈な拒否反応を抱いていた学生の一人であった。当時は必修であったプログラミングの授業履修後、自力でプログラムを書くことはできなかった。

しかし、コンピューターを主体的に使えるようになりたい、という思いはあった。そこで、情報教育をテーマとする大岩研究会１を履修することにした。研究会を履修するに至った動機としては、2 つあった。１つは、情報教育をテーマとしている研究会であるので、プログラミングに苦手意識を抱く私でもレベルに合わせて学んでいくことができるのではないかということ、そしてもう１つは、プログラミングに対して苦手意識を抱く立場から、実際の情報処理の授業を考えていきたいと思ったことである。

研究会を履修してすぐ、私は ToonTalk を使った子供向けパソコン教室のカリキュラム作成に取り組んだ。このカリキュラム作成を通じて、私はプログラミングの重要性に気づくこととなった。

プログラム作成を通じてではなくプログラミングの重要性に気づいたことは、プログラミングの本質が‘ものをつくる’上で共通したプロセスであることを証明することとなる。

1.2.ToonTalk とは？

KennKahn 氏がアメリカの子供向けに作成したプログラミングソフトである。私が参加したパソコン教室ではその日本語版を利用した。全体がパズルモードとフリーモードで構成されており、パズルモードでは、マーティというキャラクターのヒントをもとにパズルを解いていく。パズルの難易度が上がるに従って、こちらが与えた命令を実行するロボットや一つの動作を何度も繰り返す杖といった様々な道具が登場する。フリーモードでは、パズルモードで出てきた道具を使って自分で自由にゲームをつくることができる。パソコン教室は、2 日間の日程で、1 日目にパズルモードをし、2 日目にはフリーモードで自由にゲームをつくるという内容であった。そして、私がカリキュラム作成をしたのは、1 日目にあたるパズルモードを使っての授業である。

1.3.カリキュラム作成

1.3.1. パズルモードを使った授業の最終目的

子供向けパソコン教室全体の最終目的は、《子供にパソコンを楽しんでもらう》ということであった。ゆえに、パズルも子供が解いていて分かりやすく、且つ楽しいものにする必要があった。そしてさらに、最後に子供たちが自由にゲームをつくる際に、パズルモードで身につけた道具を使ってどのようなゲームをつくることが可能なのか、あるいはつくりたいものに対して、どのように道具を使えば自分のつくりたいゲームができるのかをある程度予測

できるようになってもらいたかった。これがパズルモードを使った授業の最終目的である。
そしてこれがすなわち、プログラムを書くということにあたる。自分のつくりたい作品に対して、目的を明確化し、そのための内容を階層化することからプログラムを書くということは始まる。このパソコン教室のもう一つの最終目的として、後に子供たちに、「目的を明確化し、内容の構造をきちんと練ってからものをつくる」、ということの重要性に気づいてもらいたい、ということがあった。

1.3.2. パズルの階層化

パズルモードを使った授業の最終目的に基づいて、パズルの階層化に取り組んだ。まず、基本的な一つ一つの道具の習得を一つのパズルの学習目標とし、そのためにどのようなことができることが必要なのか、それを学習目標達成のための構成要素とした。パズル全体としては、段階を追うに従って、先に習得した道具を組み合わせて解くものにし、難易度をだんだんと上げていくようにした。

1.3.2.1. パズルごとの学習目標の明確化

学習目標を明確にするというのは、非常に難を要する作業であった。
私が初めて作った‘学習目標と構成’は、目標と構成の関係になっておらず、目的（より大きな、最終的に目指す目標）と目標の関係になってしまっていた。すなわち道具の習得である学習目標が大きすぎたのである。
そこで、難易度の高い道具の習得に関しては、一つのパズルにしてしまうのでなく、いくつかの段階を追ったパズルをクリアしていくことで、習得できるように再度考え直すことにした。
その際私は、始めにつくった‘学習目標と構成’をもとにつくっていった。学習目標が大きすぎるパズルに関してだけ内容をいくつかの要素にかみくだいて、その一つ一つの要素を新たな学習目標とするパズルに分けていった。

1.3.2.2. インストラクション作成

学習目標と構成’をもとに、パズルの内容である、インストラクションを実際に考えてみることにした。
しかし、具体的なインストラクションをつくっていくうちに、矛盾が生じていることに気づいた。インストラクションと学習目標が対応していなかったのだ。きちんと階層化できておらず、‘学習目標と構成’がなくてもつくれるインストラクションであった。つまり、‘学習目標と構成’が適切でなかったのである。

1.3.2.3. 学習目標の練り直し

‘学習目標と構成’がきちんと階層化できていなかった原因は、インストラクション作成前に‘学習目標と構成’を練り直す際に、最初につくった学習目標の中で大きすぎるものにだけ手を加えて、部分を少し変えたにすぎず、全体の構造を考えることがなかったからである。

反省をもとに、‘学習目標と構成’をもう一度練り直すことになった。今度は全体の構造を考慮しながらであった。

‘学習目標と構成’を完成させるにあたって 1 番厄介であったのは、「爆弾」という道具を教える段階を見極めることであった。

「爆弾」は、一度自分が解きかけたパズルやつくりかけたゲームを壊して最初の状態に戻す、Delete のような機能をもつ道具である。

パズルモードの場合、解きかけてうまくいかないことが判明した場合、パズルを解く部屋を一度出てしまえば、最初の状態に戻ることができる。従って、「爆弾」が有効となるのは、後のフリーモードでゲームをつくる際である。ゆえに、「爆弾」を習得するパズルをどの段階に組み込ませればよいのかがとても判断しづらかったのである。

しかし、「爆弾」の機能は基本ルールであり、他のどの道具と同列になるかという視点で考えれば、最初に習得すべき、後に難易度の高い道具を習得する上で、基礎となる道具を習得するパズルに組み込ませるべきだということが明確になった。

1.4.カリキュラム作成から学んだこと

カリキュラム作成に関わったことで学んだことが二つある。

1 つは、実装に移る前に、きちんと階層化された構成を考えるということ、そして 2 つめは、デバックの重要さである。

1.4.1. 実装に移る前に構成を考えること

一見、実装に移る前に構成を考えることは、回り道のように感じられる。私はこの子供向けパソコン教室のカリキュラム作成に関わるまで、構成はつくっていくうちに自然とできてくるものだと考えていた。しかし、実は最初に構成を考える方が、スムーズにものをつくることができる。最初に構成を考えないと、内容を構成するはずであった要素が欠けていたり、全体がきちんと階層化されていない、他者から見て分かりにくいものができてしまう。今回のカリキュラム作成におけるように、実装の段階でつまずくのであれば、構成がきちんと論理的にできていないということであり、もう一度構成を練り直してから、実装に移ることが必要である。

1.4.2. デバックの重要性

構成が適切でないと判断した場合、一度つくった構成に手直しを加えるのではなく、もう一度最初から練り直す潔さが必要である。せっかくつくったものを壊すという作業は、もったいない気がする。だが、一部分に手を加えて体裁を整えるよりも、もう一度全体をつくり直した方が、きちんと構造化されたものができることを実感した。

つまり、‘ つくる、壊す ’の問題解決を繰り返しながら徐々に完成させていくということを学んだ。それまでの私は、とりあえずつくってみて、体裁の悪いところをつぎはぎのように取り繕って、完成としていた。一度つくったものを全体的に見直して、どこが問題点であるか分析し、最初からつくり直すということはほとんどなかった。しかし、全体がきちんと構造化されてこそ、自分が伝えたいことを欠くことなく、順序だてて伝えられる、他者から見ても分かりやすいものをつくることができるのである。

2. 研究背景

2.1. SFC でプログラミングの授業がほぼ必修とされる理由

SFC でのプログラミングの授業の目的は、コンピューターに対して主体的な姿勢を身に付けるというものである。すなわち、コンピューターをただ受身的に、問題解決の手段として使うのではなく、コンピューターに対して主体的に、プログラミングという形で自分が目的とする仕事をさせることで、問題解決できるようになる、ということである。

2.2.授業履修後の習熟度とその理由の模索

授業の目的とは裏腹に、プログラミングの授業履修後、自力でプログラムを書くことができるようになる学生の割合は極めて低い。その理由の一つとして、授業に対するモチベーションの低さ、すなわち授業履修以前に、既にプログラミングに対して苦手意識を抱いている学生が多い、ということが挙げられる。

プログラミングに対して苦手意識を抱いている学生は、授業の目的として掲げられるような、コンピューターに対する主体的な姿勢は、IT 関係の仕事に就かない限り必要ないと考える。それは、プログラミングといった場合に、専門的な知識を必要としそうな文章をイメージとして思い浮かべるからではないだろうか。すなわち、プログラミングに対して苦手意識を抱いている学生は、プログラミングの本質が、プログラムを書く上での文法を習得することにあると認識している。

2.3.プログラミングの定義と学ぶことの意義

プログラミングとは、「行動計画」を立て、実装し、評価、再構築を繰り返すプロセスを指す。従って論理的思考力や問題解決力が求められる。

プログラミングに対して苦手意識を抱く学生は、先に述べたようなプログラミングの本質を正しく認識していない場合が多い。だからこそプログラミングが、IT 関係の仕事に就かない限り必要のないスキルであると考えてしまうのである。しかし、プログラミングの本質である「行動計画」を立て、実装し、評価、再構築を繰り返すプロセスというのは、なにもコンピューターの世界に限ったことではなく、‘ ものをつくる ’ 上で共通するプロセスである。ゆえに私はプログラミングが、多くの人にとって学ぶ意義のあるものだと考える。

3. 研究の最終目的と課題

3.1.研究の最終目的

学生にプログラミングを通じて論理的思考力や問題解決力を身につけてもらうこと

3.2.最終目的達成のための研究課題

① 学生がプログラミングに対して苦手意識を抱く原因を明確にすること
② ①で明確になった苦手意識の原因に対して具体的な対処方法を提示すること
③ プログラミングに対して苦手意識を抱いている学生も楽しむことのできる授業を提案すること

3.3.研究課題からみた本文の構成

ToonTalk を使った子供向けパソコン教室でのカリキュラム作成を通じて、私はものづくりのプロセスがプログラミングの本質である、計画・実装・評価・再構築のプロセスそのものであることを学んだ。と同時に、プログラミングの重要性を感じた。

このことから私は、プログラミングに対して学生が苦手意識を抱く原因として、プログラミングの本質を理解していないことがあるのではないか、という仮説を立てた。

その仮説についての検証を 4 章でおこない、プログラミングに対して学生が苦手意識を抱く原因を明確にする。そして、5 章では、苦手意識の原因に対して具体的な対処方法を提示し、そのような学生でも楽しむことのできる授業を提案する。

4. Squeak でのプログラミングを通じての苦手意識の原因明確化

4.1.プログラミングに対する苦手意識の原因についての仮説

ToonTalk を使った子供向けパソコン教室でのカリキュラム作成を通じて、プログラミングに対して学生が苦手意識を抱く原因として、「プログラミングの本質を理解していないことがあるのではないか」、という仮説を立てた。

私は、プログラミング初心者にとってプログラミングの本質は理解しにくいと思う。その原因として考えられるのが、プログラムを作成する際に、文法の知識が必要とされることである。例えば Java などのプログラミングソフトの場合、プログラミング言語特有の文法があり、それを習得しなければ実装に移ることができない。ゆえに、文法を頭に入れて使いこなせるようになることが、すなわちプログラミングの本質であるかのような誤解を生みやすい。

そこで仮説を検証するにあたって、プログラミング言語特有の文法を習得する必要が少ないプログラミング環境で、プログラミング初心者がプログラム作成に取り組んだ場合、プログラミングの本質を理解することができるのかどうかというテストを行うことにした。なお、被験者は、プログラミング初心者である私自身である。

4.2.仮説検証のためのテスト

4.2.1. 利用したプログラミング環境

Squeak というプログラミングソフトを利用した。Squeak のプログラミング環境では、プログラミング言語特有の文法を習得することに時間と労力を費やす必要がない。

スクリプトは、日本語で命令の書いてあるタイルを組み合わせてつくる。さらに Squeak ではその文法の正誤を、スクリプトにタイルが組み込まれるか組み込まれないかで表現するので、プログラムをある程度書き終わった後に、実行してみて動かないということはないことになる。

4.2.1.1. Squeak とは？

教育目的で開発されたオブジェクト指向プログラミング環境である。今回私は、SmallTalk ベースのヴィジュアルプログラミング環境としての SqueakToys を利用して、シミュレータの作成に取り組んだ。(本文中 Squeak プログラミングと表記した場合、SqueakToys を使ったプログラミングのことを指す)

4.2.2. プログラムを書いて作成したものとその意図

シミュレータを作成することにした。その意図としては、2 つある。1 つは、Squeak と

いうプログラミング環境に早急に慣れるためである。そしてもう 1 つは、シミュレータ作成が、現実の現象を抽象的な現象に置き換え、制御する、というプログラミングに欠かせない作業を中核としているからである。

4.2.2.1. シミュレータの対象

シミュレータの対象は、SCM(サプライチェーン・マネジメント)である。まず、対象とした事象を詳しく調べることから始め、実際に作成するシミュレータ像を考えていった。

シミュレータの対象決定までの経緯

当初は、MD のランダム機能のシミュレータをつくろうと思った。
しかし、私はプログラミング初心者のため、自分が興味のある分野、比較的詳しいもののシミュレータをつくる方がよいとのアドバイスを受けた。

というのも、MD のランダム機能というのは、実際に曲目の順番を入れ替えるわけではなく、機械が目次を参照して音を出力するという決まった型の動作(仕事)であるため、応用性が望めないということからであった。

そこで、実体験に基づく、応用性のあるような題材を探すことにした。私は今、アパレル関係のアルバイトをしていて、在庫整理、商品搬入・戻しに効率の悪さを感じた。そこで、最近物流業界の主流となっている SCM について詳しく調べ、シミュレータをつくってみたいと思った。

SCM について、曖昧な知識しかなかったので、きちんと調べたいと思ったこと、実際に自分の体験で感じた問題点の解決策を見出したいと思ったことがシミュレータ作成にあたっての一番の動機であった。

SCM とは？

従来物流業界において、在庫を多くそろえておいて、顧客のニーズに対応していく手法がとられていた。しかし、在庫を多く抱えていると、利益率や回転率が悪くなってしまう。そこで、物流における業者間の障壁をなくし、商品搬入のスピードを上げることで、いかに在庫を少なくして多様化する顧客のニーズに応えていくか、ということで考案されたのが SCM(サプライチェーン・マネジメント)である。

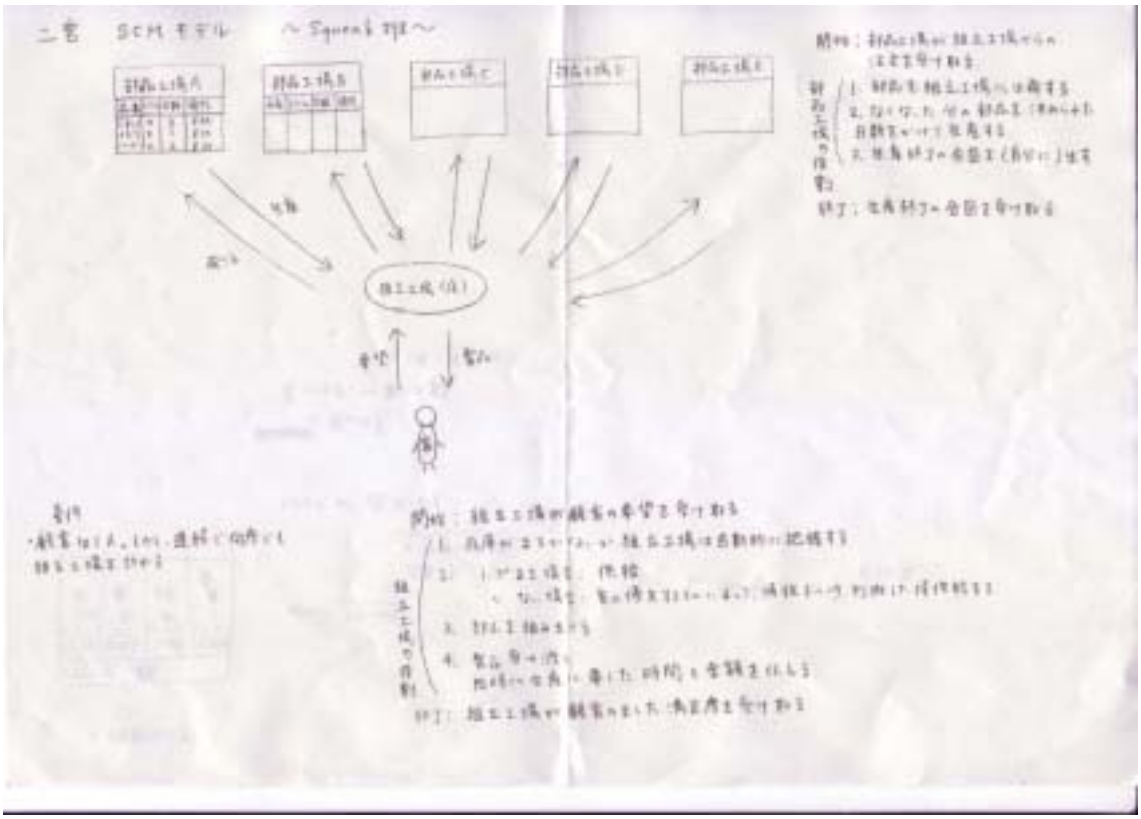
実際に作成するシミュレータ像の具体化

売買する商品をパソコンとし、売買形態としては、既存の完成商品を販売するのではなく、顧客がパソコンの部品ごとに注文し、その注文を承った組立工場が部品工場から部品を仕入れ、組立工場が仕入れた部品を組み立てたものを顧客が購入する、というものである。

登場人物は３つで、顧客と組立工場(店)と部品工場である。顧客と組立工場は一つずつで

あるが、部品工場は複数ある。部品工場によって製造可能な部品の品質レベルが異なり、さらにそれぞれが価格と製造にかかる日数のパラメータをもつ。部品工場には常に在庫があるとは限らない。ゆえに顧客は部品を注文する際、部品の品質レベルに加えて、それぞれのパラメータについても優先順位を告げる。顧客の注文を受けた組立工場は、まず部品工場の在庫を確認し、ある場合は仕入れをし、ない場合は、パラメータに対する顧客の優先順位と比較して、選択された部品工場から部品を仕入れる。

以下に、私が紙に書いたシミュレータ像を載せておく。



4.2.3. シミュレータ作成の目的

従来組立工場の人が、各部品工場を、価格と製造にかかる日数などをパラメータとし、顧客の要望と比較して、選択した部品工場から部品を仕入れていたのを、機械が、入力された顧客の要望を判断することで、部品工場を選択し、組立工場に仕入れるようになったことを視覚的に示すことである。

4.2.4. シミュレータ作成(実装)

4.2.4.1. 実装前の演習

自分で描いた絵を動かすこと(アニメーション)をしてみた。Squeak はペイントツールが充実しているので、プログラムを書くというよりも、絵を描くことに力を注いでしまった。

4.2.4.2. 実装 1(3 つの登場人物の、基本となる流れの作成)

まず、店が既存のパソコンを工場から仕入れ、顧客が購入する、という流れをつくった。トラックを移動させて、工場から商品であるパソコンを搬入し、店はそれを仕入れる。そして顧客がそれを購入する、というプログラムを書いた。

4.2.4.3. 実装 2(作成するシミュレータの構造を具体的に練る)

作品の完成に向けて、自分がつくりたい SCM シミュレータを具体的に紙に書き表してみた。それぞれの登場人物の役割を、順を追って細かく書いた。

4.2.4.4. 実装 3(製造にかかる日数のパラメータの導入)

部品工場において、製造にかかる日数の少ない部品から、組立工場に対する仕入れとして出庫していく、という動き(仕事)をつくった。このとき、最終的な SCM シミュレータでは部品工場が複数あるとしていたが、まずは簡略化し、一つの部品工場で、製造にかかる日数の少ない部品から出庫する、という動き(仕事)をつくることにした。

4.3.シミュレータ作成をしてみての考察

4.3.1. Squeak のプログラミング環境について

私は Squeak のプログラミング環境に慣れるまでにけっこう時間がかかった。最初に十分なある程度の時間集中して Squeak をしてみることが必要だと思った。

既存の日本語で命令の書かれたタイルを組み合わせていくことで、プログラムが書けることは、とても親しみやすかった。条件分岐も‘テスト’というタイルをベースとして使えばでき、分かりやすかった。

また、Java などで複合的な動きをつくろうと思うと、文法を覚えなければならないため、ある程度の時間を要するが、Squeak では、複数の動き(仕事)を関連付けた作品が比較的短時間でできるので、達成感があった。

4.3.2. シミュレータ作成について

実装 1 で、ひとまず作品をつくることにこだわってしまい、最初にシミュレータの全体像を考えることなく、実装へと移ってしまった。

しかし、最終的な作品をつくり上げていくにあたって、まず最初は、作品をつくることにこだわるのではなく、全体像をどうするかを考えるべきであった。

また、これはシミュレータをつくるということ全般にいえることであるが、どこまで実際の物流システムを簡略化して、作品としてのシミュレータをつくるかということ、そしてそれがどのような単純な動き(仕事)から構成されるか、今回の場合 Squeak のどの機能を使えばよいのか大体の見当などをつけることに先に取り組んでから実装に移るべきである。

実装 2 で、シミュレータの構造を具体的に紙に書き表すことで、何が基盤となる動き(仕事)で、何が応用となる動き(仕事)なのかがかなりはっきりとしてきた。その結果、実装に極めて取り組みやすくなった。まずつくりたい作品の構造を具体的に紙に書いてみる、という

作業の重要性を実感した。

4.4.プログラミングに対する苦手意識の原因についての仮設の検証

4.4.1. 仮説の再確認と裏付け

4.1 で、私は「学生がプログラミングに対して苦手意識を抱く原因として、プログラミングの本質を理解していないことがあるのではないか」、という仮説を立てた。

そしてその裏付けとして、「プログラミング言語特有の文法が、プログラミングの本質をみえにくくしているのではないか」とした。

4.4.2. 仮説の検証

Squeak では、日本語で命令の書かれたタイルを組み合わせてスクリプトを組み立てるので、Java のようなプログラミングソフト特有の文法を習得する必要はない。

Squeak でのシミュレータ作成において、一番労力を注ぐことになったのは、実装に移る前に、自分がつくりたいシミュレータの構造を具体化することであった。ここでの具体化とは、複雑なシミュレータの動き(仕事)を単純な一つ一つの動き(仕事)にかみくだき、さらに階層化して組み立てることを指す。すなわち、プログラミングの本質である。

Squeak プログラミング環境では、自分がつくりたいものの構造を具体化しないと、実装がうまくいかないことにすぐに気づく。このことは、4.3.2 における、私の実装での反省をみても明らかである。

以上のことから、「プログラミング言語特有の文法が、プログラミングの本質をみえにくくし、その結果学生がプログラミングに対して苦手意識を抱いてしまう」、という仮説は真であるといえる。

5. MindStorms を利用したプログラミング授業の提案

5.1. 苦手意識の原因に対する対処方法の提示

Squeak でのプログラミングを通じて、プログラミングに対する苦手意識の原因が明確となった。それは、「プログラミング言語特有の文法が、プログラミングの本質をみえにくくし、その結果プログラミングの本質が理解しにくい」、ということであった。

以下に、苦手意識の原因となっている 2 つの要素、プログラミング言語特有の文法とプログラミングの本質が理解されていないこと、それぞれについての対処方法を提示する。

5.1.1. プログラミング言語特有の文法を習得する必要の少ないプログラミング環境

文法の習得にとらわれて、プログラミングの本質がみえにくくなってしまわないよう、苦手意識を抱いている学生を対象とするプログラミングの授業において使用するプログラミング環境は、文法を習得する必要が少ない、かつビジュアル的に親しみやすいものとする。

今回提案する授業では、MindStorms というプログラミング環境を利用する。MindStorms に付属する子供向けの教育用開発ツール「ROBOLAB」(以下、MindStorms プログラミングと表記した場合、ROBOLAB を使ったプログラミングのことを指す)は、道具のアイコンを線でつなぐことで、プログラムを書くことのできる、ビジュアル的に親しみやすい環境である。さらに、書いたプログラムをコンパイルして、ハードに転送することで、プログラムの実行を目に見える形で楽しむことができる。

5.1.2. プログラミングの本質が理解しやすい授業内容

つくりたい作品に対して、その具体的な構造をきちんと練ることに労力を要することが求められるような内容とする。構造をきちんと練った上で実装に取り組み、評価する。そしてうまく実行されなかった場合は、再構築を行う。そのような問題解決のプロセスも重要視する。

作品の構造化がきちんとできているかは、発表やグループ・ワークといった形で、他者に伝えることができるか、という点をポイントとして評価する。

また、問題解決についても、自ら取り組んだ上で、さらに他者とディスカッションを行うことで、何が問題であるのかを突き詰めていき、その解決策について最良の方法を探ってもらうこととする。

5.2. 苦手意識を抱く学生も楽しむことのできるプログラミング授業の提案

以下のドキュメントは、2002 年秋学期大岩研究会 1 において、履修者である佐藤聖、高

橋貴明、二宮温子が共同で作成した最終レポートの一部を抜粋したものである。

なお、2002 年度秋学期研究会最終レポートの全容については、以下を参照していただきたい。(http://web.sfc.keio.ac.jp/~t00755an/sotsuron/MindStorms.pdf)

5.3. プログラミング入門(Ｌコース)授業企画書

5.3.1. コースの位置付け

- SFC の「プログラミング入門」の１コースで、プログラミングツールの一つである LEGO Mind Storms を取っ掛かりに、プログラミングの意義を考えてもらうコース

5.3.2. 対象（Target）

- プログラミングを学ぶ事に意義を感じない人

⇒ 必要性を認識しなければ、授業の理解は進まない

5.3.3. 授業の目標（Goal）

- 『プログラミング』の重要性を理解すること

⇒ プログラミングとは、

「行動計画」を立て、実践し、評価、再構築を繰り返すプロセスを指す

5.3.3.1. 良い行動計画とは何か

- 客観的な構造を持っていること

⇒自分にとって分かりやすいだけでなく、外部評価に耐え得る

- 論理的な構造を持っていること

⇒目的の階層構造と、それを達成する手段が、逐次的に明文化されている

5.3.3.2. 良い行動計画を立てることの重要性

- 悪い計画を立ててしまうと・・・
  - 方向性を見失ってしまう
    - ⇒ 到達目標（Goal）をはっきり決めていないことに起因する
  - 効率が悪い
    - ⇒ 回り道が多くなり、最短経路を辿れない
  - 成果（成功・失敗）を評価する事ができない
    - ✧ なぜ失敗したのかがわからない
      - ⇒失敗を繰り返す
    - ✧ なぜ成功したのかがわからない
      - ⇒二度と成功しないかもしれない

5.4. Ｌコースの Objectives

- 目標達成のための的確な行動計画を立てられる
  - 要求分析ができる

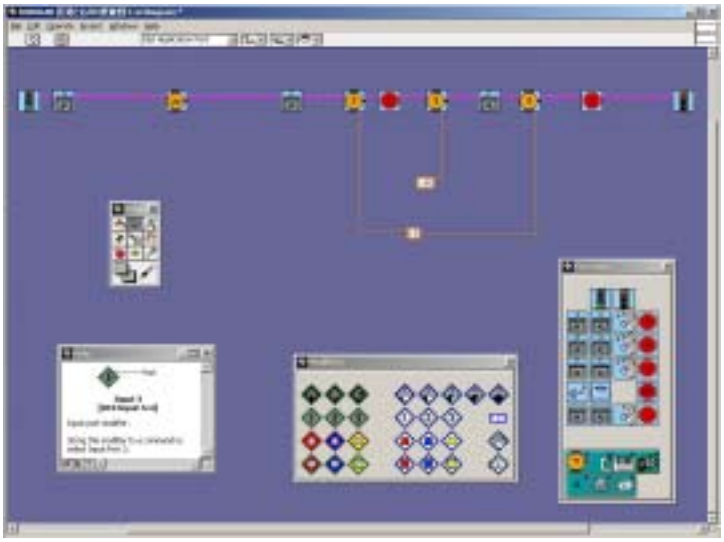
- 要求を満たす目標を設定できる
- 論理的な計画を立てられる
- シミュレーション（プレテスト：自己評価）ができる
- 計画の致命的な誤りを修正できる（デバッグができる）

- 行動計画の最適化ができる
  - 成果のパフォーマンス（どれだけ意図どおりだったか）を測定できる
    - ✧ 要した経済的，時間的，人的コストを計算できる
    - ✧ 得られた利益を客観的（比較可能な形）に算出できるシミュレーション（プレテスト：自己評価）ができる
  - パフォーマンスを最大化するような計画の再構築ができる

5.5.授業で使用するプログラミングツール

第１回～第７回までは、LEGO Mind Storms に付属する教育用開発ツール「ROBOLAB」を用い、第８回以降は、Java をベースにした開発環境「leJOS」を用いる。

5.5.1. ROBOLAB



Microsoft Windows 上のヴィジュアル開発環境。

マウス操作で、RCX（Mind Storms のコントロールセンター）の機能を割り当てたアイコンを画面上に配置し、プログラムフローに従って、アイコン間に線を繋ぐことで、Mind Storms ハードウェアの制御を実現する。

視覚的に取っ付きやすい、というメリットを考えて、コースの前半では、この ROBOLAB を用いて Mind Storms プログラミングを行う。しかし、基本的に、子どもたちに使いやすいように簡便化してあるので、画面サイズの制約、メソッドや抽象的オブジェクトが作れない、などの問題点がある。

5.5.2. leJOS –Java for the RCX-

「leJOS」は、RCX に搭載可能な Java VM で、RCX の LEGO 純正ファームウェアを書き換えて機能する。ユーザは Java でソースコードを記述し、パソコンでコンパイル、RCX に転送する。Windows、UNIX とも動作可能である。

メモリ容量の制限はあるが、プログラミング上の抽象概念なども実装できる。ただし、敷居が高いので、コースでは、後半からこの開発環境を用いる。

URL: http://lejos.sourceforge.net/

5.6. パイロットテストとしての授業の実施

提案した授業について、研究会履修者に対して、研究会の授業時間内のうち、指定された 30 分間で、パイロットテストとして実際に授業を行った。ここでは、私が担当した回の授業計画書を載せておく。

5.7.第二回授業計画書

対象(Target)　：SFC でプログラミングが苦手な人（基本的には１年生初心者）

目標(Goal)　：『プログラミング』の過程を楽しめるようになる

- ✧ プログラミング作業に入る前に「目標達成のための計画」をたてられる
  - ● 目標達成に必要な手段を考えられる
- ✧ 「計画」の良し悪しについて客観的に評価できる
  - ● 手段を最適化・効率化できる

到達目標(Objectives)　：１、単純な動作を「計画」できる

２、失敗した時に部分的に計画の修正（デバッグ）ができる

授業との対応関係　：１、前半部分（曲がり方確認～普通の駐車）

- ✧ 単純な動作を「計画」できるを主目標として
  - ● Mind Storms の概要を再説明
  - ● 「計画」を立てることに慣れる為の補助として手順を分解・明示する

２、後半部分（ワークショップ形式）

- ✧ 失敗した時に部分的に計画の修正ができるを主目標として
  - ● 自分達で１から「計画」をたてさせる
  - ● 限られた時間の中で、効率的な計画を意識させる
- ✧ 最後の自分達の計画、成果、今後の指針を発表してもらい「計画」の客観的な良し悪しについて考えさせる

今後の展開例　：１、ソフト面

- ✧ 単純な計画からより複雑な計画へ（制御構造、メソッド化等）

２、ハード面

- ✧ 車形態から、よりユニークな動作が可能な形態へ

5.8. パイロットテスト実施後の結果分析

5.9.第二回授業

5.9.1. 授業の流れ

受講者２人に一台の ROBOLAB を起動したラップトップ PC と、LEGO カーを用意。あらかじめ黒板に、前回の復習である曲がり方の確認について図を書いておき、復習は簡単に済ませた。しかし受講者の中から前回復習部分についての質問が多数出た為、急遽それらのフォローのためにカリキュラムを変更することになり、後半部分では「駐車～方向変換」を各班にプログラミングしてもらい、出来た班の代表に、それを達成するためにとった方略（計画）を発表してもらい、計画の妥当性について簡単に議論をしてもらった。

5.9.2. 反省点

- 前回授業の復習にはそれほど重きを置いていなかったが、実際には前回内容を覚えていない人がほとんどで、そこに多くの時間を費やす事になってしまった
- 当初予定していた、受講者自身に一つの動作を作る目標を立ててもらい、それについての議論をするというプロセスが、全く実行できなかった。結果、「計画」の重要性にまで議論を発展できなかった。

5.9.3. 受講者評価シートからの問題提起

- 授業時間の配分は、プレテストを実施することである程度見通しが立つのではないか。
- プログラム上の概念と実際の車の動きとの関連を、もっと強調して説明した方がよい。
- まず言葉で説明されてそれを形にしていく形式だったが、最初に完成品を実際に見せてくれた方が、イメージが湧きやすいのではないか。
- 前回と今回の授業は、全体の講義の中でどう位置付けられているのか。

5.9.4. 受講者評価シートからの評価点

- 車という題材で動作を説明した点がわかりやすかった。
- 複合的な動き（駐車）は案外難しく、きちんと考えなくてはダメだということがわかった。
- プログラミングの過程がきちんと楽しめた。

5.9.5. 次回授業への課題

- プレテストの実施、あるいはもう少し時間を掛けてタスクの分析をする等して、やってもらいたいことを時間内に不足なく実施できるようにする。
- 受講者の理解度を見ながら授業を進める。復習が必要ならば、効率的な実習も考えておく必要がある。
- 最初に完成品を見せるかどうか、メリット・デメリットを検討し、方針を決める。
- 授業の位置付け、全体の目標等をはっきりさせるためにシラバス・企画書を作成する。

6. 総括

6.1. 研究結果の総括

6.1.1. 研究課題

「学生にプログラミングを通じて論理的思考力や問題解決力を身につけてもらうこと」という研究の最終目的達成のために、以下の 3 つを研究課題とした。

① 学生がプログラミングに対して苦手意識を抱く原因を明確にすること
② ①で明確になった苦手意識の原因に対して具体的な対処方法を提示すること
③ プログラミングに対して苦手意識を抱いている学生も楽しむことのできる授業を提案すること

6.1.2. 研究結果

6.1.2.1. 苦手意識の原因の明確化

ToonTalk でのカリキュラム作成を通じて、苦手意識の原因についての仮説を立てた。それは、「学生がプログラミングに対して苦手意識を抱く原因として、プログラミングの本質を理解していないことがあるのではないか」、というものであり、その裏付けとして、「プログラミング言語特有の文法が、プログラミングの本質をみえにくくしているのではないか」とした。

これら仮説の検証は、プログラミング言語特有の文法を習得する必要が少ないプログラミング環境、Squeak でのプログラミングを通しておこなった。

その結果、Squeak でのプログラム作成において、一番労力を注ぐことになったのは、実装に移る前に、自分がつくりたいシミュレータの構造を具体化することであり、プログラミングの本質を学ぶことができることが実証された。

従って、仮説は真であることが分かり、学生がプログラミングに対して苦手意識を抱く原因が、「プログラミング言語特有の文法が、プログラミングの本質をみえにくくし、その結果学生はプログラミングの本質を理解しにくい」、ということであるのが分かった。

6.1.2.2. 苦手意識の原因に対する対処方法の提示

Squeak でのプログラミングを通じて、プログラミングに対する苦手意識の原因が明確となった。苦手意識の原因を構成する 2 つの要素、プログラミング言語特有の文法とプログラミングの本質が理解されていないこと、それぞれについて対処方法を提示した。

プログラミング言語特有の文法に対する対処方法として、苦手意識を抱いている学生を対象とするプログラミングの授業において使用するプログラミング環境は、文法を習得する必要が少ない、かつビジュアル的に親しみやすいものとすることを提案した。

そして、プログラムの本質が理解されていないことに対する対処方法としては、授業内容についての提案をおこなった。その授業内容とは、つくりたい作品に対して、具体的な構造をきちんと練ることに労力を要することが求められるようなものである。

なお、作品の構造化がきちんとできているかは、発表やグループ・ワークといった形で、他者に伝えることができるか、という点をポイントとして評価することとした。

6.1.2.3. 苦手意識を抱く学生も楽しむことのできるプログラミング授業の提案

苦手意識の原因に対する対処方法をもとに、プログラミング授業を提案した。

その際、プログラミング環境としては、道具のアイコンを線でつなぐことで、プログラムを書くことができ、ビジュアル的に親しみやすいというメリットをもつ、MindStorms を利用した。

授業内容としては、個人での要求分析や作品の構造化に加えて、発表やグループ・ワークを取り入れ、議論をおこなうことで、さらに論理的思考を養うことのできるようなものにした。

6.2. 研究全体を通じての考察

6.2.1. プログラミングに苦手意識を抱く学生の誤解

プログラミングが苦手な人というのは、得意な人を見て、プログラムというのは、最初からコンピュータが読み取ることのできる形で書けるのだと見誤ってしまう。しかし、それは本当はそうではなく、得意な人は、頭の中でプログラムの対象となる動き(仕事)を、単純な一つ一つの動きから成る、順序立てて段階を追ったものに直して、それをプログラムとして表現しているにすぎないのだ。もしくはプログラムを書くことへの慣れから、一つ一つの動作を組み立てるということを意識することなくプログラムが書けるということなのであろう。

しかし、そのような誤解を生んでしまう原因として、プログラミングの授業内容に問題があるのかもしれない。プログラミングの授業では、教える順序に気を配る必要がある。まず、プログラムをみせて、一つ一つの定義や文章について、どのような意味をもつのか、その解説から入ると、先に述べたような誤解が生じやすい。

プログラミングの本質は、コンピュータにさせたい動き(仕事)を一つ一つの単純な動きにかみくだいて、それを組み立てることである。ゆえに、プログラミング言語特有の文法はあくまで手段である。日本語で、コンピュータにさせたい動き(仕事)を、まず、一つ一つの単純な動きを順序立てた、段階をふんだ設計図として書くことが、プログラミングの授業として最初になすべきことであるように思う。

今回私が研究を進めていくにあたって参考文献とした、『Pascal プログラミング対話』がプログラミング初心者である私にとって非常に理解しやすかったのは、まさにその順序が正しかったことにある。次の付録の項で、詳しく文献の紹介をするが、『Pascal プログラミング対話』では、対話という形をとって、まずコンピュータにさせたい動き(仕事)を一つ一つの単純な動きにかみくだくことを最初に行う。それからコンピュータが読み取れる形のプログラムを書いていく、という形式が終始一貫されている。極端な話、プログラム作成にあたって、文法に精通している必要はない。コンピュータにさせたい動き(仕事)を一つ一つの単

純な動きにかみくだいて、それを順序立てて組み立てるということができれば、後は、日本語の文章をコンピュータが読み取ることのできるプログラムに書き換えるプログラムさえもあるほどである。

6.2.2. プログラミングはものづくりの手順と同じである

プログラミングが重要とされるのは、これまで述べてきた設計図が書けるようになることが重要だからである。私は、プログラムというのは文法を覚えて慣れで書けるようになるものだと思っていたから、とにかく見よう見まねで、最初からコンピュータの読み取れる形で書くことを先決としていた。しかし、そのようにしてできたプログラムは、動かないことが多く、あるいは動いたとしても、汎用性に欠けるものであったり、何が抜け落ちているのかが自分で分からないものであったりする。しかし、最初に設計図を書くことで、まずつくろうとしている対象が明確になり、かみくだいて考えることで、1 つの動き(仕事)を構成する単純な一つ一つの動きを抜かすことなく組み立てて、完成させることができる。

プログラミングというのは、ものをつくる時に計画を立て、実行することと同じなのだ。大岩研究会での活動を通じて、私はまず計画を立ててから、実装に移るということを身につけた。以前の私は、とりあえずなんとなく形にしてみて、それから手直しを加えて完成させる方が早いと思っていた。しかし、まず全体の構成を考えて、それから部分ごとにつくっていく方が効率がよく、さらにうまく動かなかった時に、どこに問題があるのかが発見しやすいということを学んだ。最初に全体の構成を考えることで、自分がつくろうとしている作品を構成する要素を抜かしてしまうこともなくなる。

最初に計画を立て、それから実装に移るというのは、プログラムの世界に限られたことではない。ものをつくるということに関して全てあてはまる。

プログラミングは、論理的思考、問題解決能力を養う上で優秀な手段である。特に、SFC の場合、様々な授業でケース･スタディーやディスカッションを行うが、その成果がダイレクトに返ってくることは少ない。しかし、プログラミングでは、計画を立て、実装し、評価、再構築というプロセスが実際に目に見える形で、ダイレクトに成果が返ってくる。そういった手段は他に少ないのではないか。目に見える形で成果が返ってくることによって、問題の存在も明らかになり、従って問題分析、再構築のプロセスを繰り返し行なうようになる。それこそが、最も重要なプロセスなのである。

付録：自習教材としての図書の紹介

私が研究を進めていくにあたって参考文献とした図書について紹介したいと思う。プログラミング初心者であってもプログラミングの本質を理解しやすく、自習教材として利用することをお薦めする。

以下、図書について簡単に、良い点と考慮すべき点を述べる。

『Pascal プログラミング対話』　森口繁一・小林光夫・武市正人共著

良い点

◇　プログラムを書く前に、自分がつくりたい動き(仕事)をかみくだき、どのような動きを組み立てていくことでつくれるのか、また組み立てる順番はどうするか、などプログラムのコメントに値する部分を丁寧につくっているところ。
プログラムを書く、ということはプログラミングソフト特有の文法が使い慣らせるようになる、ということを指すのではなく、計画・実装・評価・再構築のプロセスを指すことが明確に示されている。ゆえに、プログラミング初心者にとってプログラミングとは何であるかを理解しやすい。

◇　対話形式でプログラム作成を進めていくところ。
先生と生徒二人という設定で、プログラムでつくりたい動き(仕事)がどのような単純な動きから成っているのかを一つ一つかみくだいていく。そして、間違いが起こりやすい箇所については、必ずその間違いとなぜ正しくないかその理由が述べられており、そして正しい回答へと導かれるようになっている。また、同じことが実行されるプログラムであっても、より簡潔で分かりやすいプログラムへの改善も随所で行われている。

◇　各項の最後に練習問題が付属されているところ。
単に文章を読み進めていくだけでなく、読者自らも問題を解くことで、本文で述べられているようなプログラムが書けるようになる。また、基礎固めの段階では、特にポイントとなる問題に関しては、次項の最初の単元で、詳しく解説がなされている。

◇　付録として Pascal 文法の手引きがあるところ。
本文を読み進めていくに従って、最初の方で習った文法を応用したり、それらを組み合わせて解くような問題に取り組むようになる。従って、文法についての認識が曖昧であったら、巻末の付録で確認すればよい。

考慮が必要な点

◇ **練習問題の解答において、一部プログラムの名前が、そのプログラムの目的を表すものではないところ。**

練習問題の解答において、プログラムの名前が‘ mondai11 ’などとなっているものがある。これは、練習問題何番の解答である、ということを分かりやすく示すためであろうと思われるが、本来プログラムの名前は、そのプログラムの目的をつける。特に長いプログラムを書く際など、書いていくうちにプログラム自体の目的が何であったかが曖昧となってくることがあるので、気をつけるべきである。プログラミング初心者は特に注意すべき点であり、プログラムの名前は、そのプログラムの目的を表すものにするよう心がけた方がよい。

◇ **プログラムでつくるものが数学的なものが多いところ。**

この点は、高校で数学をあまり履修しなかった人にとっては少々つらい点かもしれない。ただ、本文中で詳しく式や解の計算方法について述べてあるので、内容に詳しく触れたことがなくとも数学に興味のある人なら楽しんで学習できると思う。プログラムでつくる部分についての解説は詳しくなされているので、プログラミングの学習を進めていく、という上では、本質的には数学の知識は関係がない。ただ、プログラムを書く際の題材として数学が取り上げられているということにすぎない。

数学が話題となっているということの利点は、解が一つに定まっているので、プログラムがうまく実行されているかが判断しやすいということであろう。プログラムを書いていく上で、デバックというのは非常に重要になってくる。数学が話題であれば、プログラム初心者であっても、プログラム実行結果の正解に大体の見当をつけることができる。正解の見当をつけることができれば、うまくプログラムが実行されなかった場合に、どこを改善すべきか考えやすいであろう。

謝辞

本研究の機会を与えていただき、ご指導を賜りました、慶應義塾大学環境情報学部　大岩元教授に深く感謝致します。

研究会での活動、今回の卒業制作でも構成を考えるにあたって多大なる貢献をしてくれた総合政策学部 4 年　佐藤聖氏、共にプログラミング授業の提案をした総合政策学部 4 年　高橋貴明氏に感謝の意を表します。

また慶應義塾大学　大岩研究会の皆様には絶えず貴重な意見を頂きました。この場をお借りして深く感謝の念を表します。

2004/01/30

二宮　温子

参考文献

『Pascal プログラミング対話』　森口繁一・小林光夫・武市正人共著