情報教育 Vol. 2 2020
Research Report of Informatics Education Vol. 2 2020
pp.31-40

【研究論文】

プログラミング学習前に行われたプログラミングゲームの理解度とその学習後の到達度との関係分析

― アルゴロジックの利用とその学習データの活用を通じて ―

Analysis of the Relationship between the Comprehension Degree of a Programming Game in the Early Stage of Programming Learning and the Achievement Results after the Learning:

― Through the Use of ALGOLOGIC and its Learning Data ―

松本慎平†1, 加島智子†2, 山岸秀一†1
MATSUMOTO Shimpei†1, KASHIMA Tomoko†2, YAMAGISHI Shuichi†1

要旨： プログラミング教育に関するいくつかの論文では，プログラミング学習後の理解度はプログラミング学習以前の何らかの適性によって決定付けられると述べている．そこで本論文では，プログラミングの適性をその学習以前に行われるプログラミングゲームの理解度で定義し，それとプログラミング学習後の理解度との関係を調査することを目的とする．すなわち，プログラミング学習前に動機付けの目的で利用されたプログラミングゲームの理解度の状況を個々のみならず全体の傾向も含め事前状態として把握するばかりでなく，事前状態と事後状態であるプログラミング学習後の成績とを対応付けて分析することを課題とする．本論文では，事前状態を把握するためのプログラミングゲームとしてアルゴリズム思考の体験を目的とした「アルゴロジック」を利用した．アルゴロジックは，プレイヤが予め設定した命令ブロックでロボットを自動的に動作させ間接的に問題解決を行わせるプログラミング未経験者向けのパズルゲームである．本論文では，学習者がプログラミングの学習を始める前にアルゴロジックを用いた演習を行い，演習後その理解度を確かめるためのアルゴロジックテストを実施した．学習者は，アルゴロジックテストの後，プログラミングの基本を約 10 か月間かけて学習し，プログラミング学習の理解度を都度テストで確認した．本論文では，これらテストの結果をアルゴロジックテストの得点と関連付けて分析した．分析の結果，アルゴロジックテストの結果とプログラミングの理解度との間には正の関係が示唆された．

キーワード： プログラミング，テスト分析，プログラミングゲーム，アルゴロジック
Keywords： Programming, Test Analysis, Programming Game, ALGOLOGIC

1 はじめに

プログラミングはソフトウェア開発やコンピュータサイエンスに重要な技能と一般的に認識されている．大学などの情報学に関連する多くの教育機関ではプログラミングは特に重要な科目と位置付けられており，それゆえに他の科目以上にコストをかけて教育を行っている．プログラミング教授者や研究者らはその学習効果を高めるため様々な取り組みを推進しており，学術雑誌などでも成果の一部を確認できる[1)]．近年特にコンピュテーショナル・シンキング[2,3)]といった考え方が現代社会に必要な能力として注目を集めており，プログラミングの重要性は今まで以上に社会から強く認識されている．一方，プログラミング教育現場では，理解度の二極化という無視できない課題に直面する場合が多い．この二極化の問題は，一部の研究者によってプログラミング教育の重大な問題と指摘され，いくつか先行研究は二極化の原因と考えられる要因を指

†1 広島工業大学 Hiroshima Institute of Technology
†2 近畿大学 Kindai University

摘している[4),5),6),7)].

前述のとおり様々な改善がプログラミング教育に行われているにも関わらず，二極化を解決する有効な手立ては現在もなお見付かっていない．一方，二極化の解決に向けては，プログラミング導入教育で学習者に興味を持たせることが最も有効な解決策の一つと一般的に受け入れられ，またそれは教育現場でも経験的に了承されている．実際，プログラミング初心者向けの導入教育では，プログラミングの概念を伝えること，プログラミングの面白さを伝えること，プログラミングの学習に重要な要素を体験させること，といった3つの目標を設定している場合が多い[8),9),10)]．これらの目標を実現するアプローチとして，パズルのようなアルゴリズム設計を体験できるプログラミングゲームまたは Scratch などのビジュアルプログラミング言語がプログラミング導入教育で積極的に採用されている[11),12),13),14),15)]．プログラミングゲームとビジュアルプログラミング言語は，目標が事前に与えられるかどうかといった点でその特性は異なっている．一般に，プログラミングゲームは明確な目標を提供し，また問題解決の手段を制限する．一方ビジュアルプログラミング言語の場合，コーディングを容易化する手段の提供に留めているため，学習者自身が自由に目標を設定しなければならない．プログラミングゲームは目標が明確であるため，あらゆる学習者にとって取り組みやすく，そのためプログラミング講義などで利用しやすい．以上のような背景から，プログラミングゲームは学習者を動機付けながらプログラミングの基本的な考え方を短時間で伝える用途としてよく利用されている[16)]．

著者らは所属機関でプログラミング未経験者を主対象にしたプログラミング導入科目を数年間担当している．この授業の中では，学習意欲を高めること，プログラミングに必要な概念を理解させることを目的としてプログラミングゲームを活用している．著者らが利用しているプログラミングゲームは，一般社団法人電子情報技術産業協会（JEITA）が開発したアルゴリズム体験ゲーム「アルゴロジック」[1]シリーズである．アルゴロジックは，コンピュータサイエンスに興味を持つプログラミング未経験の(あるいはプログラミング経験の浅い)学習者を対象とし，プログラミングの基本となる論理的思考(アルゴリズム)をゲーム感覚で体験可能な課題解決型ゲームである．著者らは，プログラミング学習の前段階でアルゴリズム的思考に慣れることがプログラミング学習に良い影響を与えると考えており，プログラミング導入科目の初期段階でアルゴロジックを利用している．加えて，その演習後アルゴロジックでの理解度を確認するテスト(以降，アルゴロジックテスト)を実施しており，理解が不十分な学習者の支援や講義計画の参考にするために得られたデータを活用している．

本論文は，アルゴロジックテストの理解度の状況を個々のみならず全体の傾向も含め事前状態として把握し分析すると共に，事後状態であるプログラミング学習後の成績とを対応付けて分析することを目的とする．プログラミング教育にプログラミングゲームを活用した事例は多く存在するが，プログラミングゲームの理解度を学習者の分析に活用した取り組みはほとんど見られない．特に，プログラミング基礎の学習後の到達度をプログラミングゲームの理解度と紐付けて分析した取り組みは十分に行われていない．

本論文ではまず，アルゴロジックテストの得点を単独で分析した．次に，学習者がプログラミングの基本概念を学んだ後に受験する到達度テストの得点をプログラミングの理解度とみなし，アルゴロジックテストの結果と関連付けて分析した．具体的には，アルゴロジックテストを事前テスト，プログラミングの到達度テストを事後テストとし，事前テストの得点に基づき学習者を二群に分け，群ごとに事後テストの平均値を調査した．分析の結果，事前テストで良い得点を取っていた群は，事後テストでそうでなかった群と比べて得点が統計的に有意に高く，事前テストと事後テストの得点には正の関係が示唆された．この分析結果だけでは因果関係まで明らかにできないため可能性の一つに過ぎないが，プログラミングの適性，例えば読解力や構造把握力など汎用的技能が不十分か，あるいは問題解決の手順を理解することに十分な動機を持っていない学習者は，プログラミング学習が順調ではないことを示唆する傾向が確認された．

2 関連研究

昨今，プログラミング学習に対する期待や要望はこれまでにないほど高まっている．このような背景から，様々なプログラミングゲームが開発され，大学の授業や民間のスクールなどで積極的に利用されている．プ

[1] https://home.jeita.or.jp/is/highschool/algo/

ログラミングゲームとしては，古くは RoboCode，最近では Minecraft，Elevator Saga[2]が有名である．同様な理由から，ビジュアルプログラミング言語の開発や研究も進められている[17),18),19),20),21)]．

Squeak[22)]や Scratch[9),10),11),12),13)]は，ビジュアルプログラミング言語の代表である．ビジュアルプログラミング言語は文法エラーを回避できることから，アルゴリズムの構築や理解に集中できる．よって，プログラミング初学者にとって扱いやすい言語とされている．プログラミング学習と同時にアルゴリズムの構築能力の養成を目指した取り組みとして，BlockEditor[23)]や oPEN[3] [24)]がある．これらは，OpenBlocks[25)]という仕組みを用いており，従来無かったブロック型言語からテキスト型言語へのスムーズな移行を支援している．

プログラミング力を養成するためにプログラミングゲームが有効かどうかは現時点では不明である．Hour of Code[4]によると，プログラミングゲームの多くは，プログラミングの技能，創造性，問題解決力を高めることを目的に開発されていないと断っている．すなわち，コンピュータサイエンスを学ぶことがどれだけ身近で簡単で，面白いかを学習者に伝えることが狙いだと述べている．プログラミングゲームを教育に用いた事例はいくつか行われているが[8,9,16)]，Hour of Code の考えのもとで実践されたものと考えられる．

3 プログラミングゲームの活用

プログラミングゲームを実際の講義やプログラミング体験イベントに導入し，事後アンケートを通じて学習者から得られた意見や感想に基づきプログラミングゲームの有用性を明らかにした研究は多く存在する．一方，プログラミングゲームに対する理解度そのものを分析した事例や，その理解度の全体傾向の複数年にわたっての分析，プログラミングゲームの得点とプログラミング学習後の学習者の事後状態とを対応付けて分析し考察した取り組みは十分に行われていない．したがって，これらを本論文の主な取り組みとし，そのための分析方法を検討した．

[2] http://www.businessinsider.com/15-free-games-that-will-help-you-learn-how-to-code-2017-4/

[3] https://github.com/xDNCL/oPEN

[4] https://hourofcode.com/us

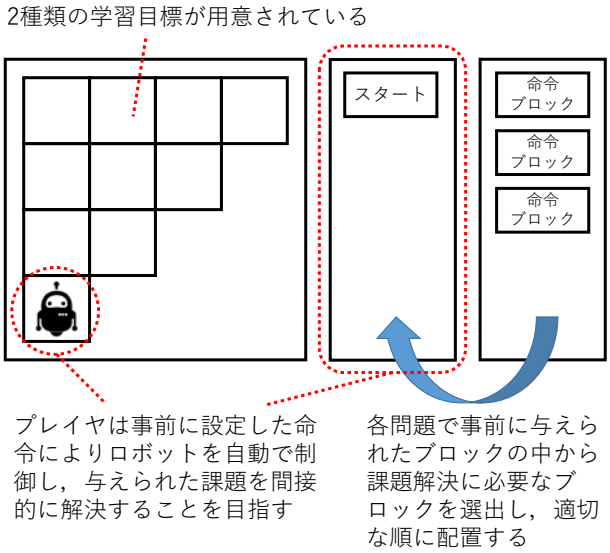


図 1 アルゴロジックの基本ルール

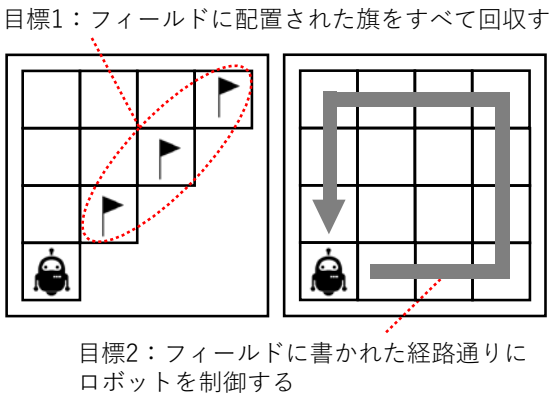
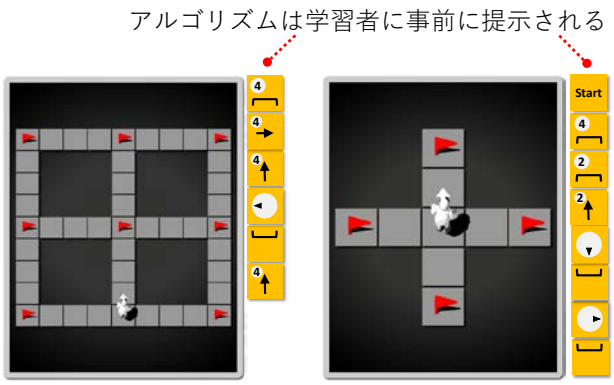


図 2 アルゴロジックの課題

3. 1 アルゴロジックのルール

アルゴロジックは，命令ブロックを適切な順番で事前に入力欄に配置し，その与えられた命令ブロックによりロボットを自律的に動作させることで与えられた課題の解決を目指すパズルゲームである．アルゴロジックの概要を図 1 に示す．学習者に提示される課題としては，フィールドに設置された旗を全て回収する課題，指示された道順のとおりロボットを動作させる課題，の 2 種類がある．アルゴロジックにある 2 種類の課題は図 2 に示すとおりである．プレイヤは，問題ごとに事前に用意された複数の命令ブロックの中からいくつかを選出し，適切な順に命令ブロックを並べる．命令によりロボットを自動的に動作させることで，そのロボットに間接的に仕事を処理させる．アルゴロジックが持つ主要な機能の詳細は以下のとおりである．

- プレイヤは，選択肢の中から命令ブロックをひとつ選び，事前に用意されたスタートブロックの下部にドラッグ&ドロップで接続する．2 回目以降は既に設置された最下部ブロックの下部にブロックを接続する．この作業を必要なだけ順に繰り返し，アルゴリズムを組み立てる．命令ブロックの種類として，前後，左右，回転，繰り返し，分岐(アルゴロジック 2 のみ)がある．プレイヤは，移動量や繰り返し回数を 1～x の整数値で設定できる．
- プレイヤは，命令ブロックの配置後スタートボタンをクリックし，ロボットの動作を確認できる．ロボットは与えられたブロックの命令通りに動作する．ロボットの動作中にスタートボタンをクリックするとロボットは一時停止する．
- 最小の命令ブロックでゲームクリアした場合，そのことが画面に表示される．最適な命令でない場合であってもゲームクリアできるが，その場合最適ではないことが画面に表示される．



学習者は与えられたアルゴリズムが適切かどうか(問題を解くことが可能かどうか)を最適性については考慮せず2択で回答する．

図 3 アルゴロジックテストで出題される問題例

3. 2 演習方式及び理解度確認方式

前述のとおり，プログラミング未経験(あるいは経験の浅い) 大学 1 年生を対象にしたプログラミングの基礎科目の序盤において，学習意欲を高めながらプログラミングの考え方を伝えるためにアルゴロジックが利用されている．アルゴロジックは 90 分間の 2 回の講義内で利用されている．このアルゴロジックを用いた演習では，学習者がアルゴロジックのルールを正しく理解できているかどうかをアンケートで確認し，全員がルールを理解できるまで教授者は個別に説明を繰り返し行っている．全学習者がルールを十分に理解できていることを確認した後，理解を確認するためのアルゴロジックテストを実施している．アルゴロジックテストは，アルゴロジックを題材とした 10 問程度の問で構成されている．各問は，白紙の状態からアルゴリズムを設計するものではなく，フィールドとアルゴリズムが事前に与えられていて，それが正しいものかどうかを応える形式である．問題の一例を図 3 に示す．問題文は次のとおりである：「ロボットが全ての旗を取ることができれば(ロボットが指示された経路に沿って動作できれば)ゲームクリアです．ここに示されたステージに対して右側のアルゴリズムは問題を解くために適切かどうか回答しなさい．なお，最適なアルゴリズムである必要はありません．」

アルゴロジックテストは Moodle の小テストモジュールを利用して学習者に提示され，学習者は二者択一の形式で各問に回答する．各小テストの制限時間は 1 問 1 分とし，10 問 10 分を基本としている．可能な限り正確なデータを得るために，私語，持ち込みを許可しない試験の形式で実施している．

表 1 平均 GPA

	前期	後期
1 年目	2.12	2.35
2 年目	2.03	2.22
3 年目	2.24	2.29

4 分析結果

4. 1 アルゴロジックテストの分析

本論文では 3 年間にわたって収集されたデータを分析対象とした．分析対象者は大学 1 年生であり，前期・後期に開講された C 言語の基礎を学ぶ必修科目の中で得られたデータを分析に用いた．1 年目の分析対象者数は n=104，2 年目は n=102，3 年目は n = 103 とした．これら分析対象者には，大学 2 年生以上の学習者や途中で授業を辞退した学習者は含まれていない．まず，前期に開講された講義の中でアルゴロジックテストを実施し回答を得た．本論文では，アルゴロジックテストの得点をプログラミング学習前の事前状態と位置付けた．アルゴロジック演習の後，前期と後期の授業の中で，数回プログラミング理解度確

認テスト(以降, プログラミングテスト)を毎年行っている. 本論文では, これらプログラミングテストの得点から学習者の学習後の状態を把握可能と仮定し, プログラミングテストの得点を事後状態と位置付けて分析に用いた. なお, 学習前後の状態の関係を分析するため, 前期と後期の両方のクラスを受講し, 十分な出席を行い単位評価の権利を得た学習者のみを分析対象とした. プログラミングテストは3回あり, 1回目を試験 1, 2 回目を試験 2, 3 回目を試験 3 とした.

3 年間の全ての講義は同一の教授者が授業を担当し, 授業やテストの内容, 課題の数, テストの難易度は全て同程度であった. 学習者の特徴や授業の難易度の均一性を確認するため, 平均 GPA を表 1 のとおり算出した. 平均 GPA とは, 受講者の到達度(グレードポイント)を 0 から 4 の 5 段階で評価し, 試験が未受験であった場合や出席回数が基準に満たないなど単位評価の権利を得ない学習者を受講放棄者としたとき, 受講放棄者を除く全学習者の到達度の平均値である. 表 1 で示されるとおり各年の間には多少の差が見られるが, Welch の t 検定では群間に有意な差は確認されなかった. 以上から, 3 群のプログラミング力はほぼ同程度と見なせることとした. 課題の提出状況についても, 3 群の間に大きな違いは見られなかった. 以上から, 学習態度や学力水準に大きな乖離は存在していなかったと見なすことができる.

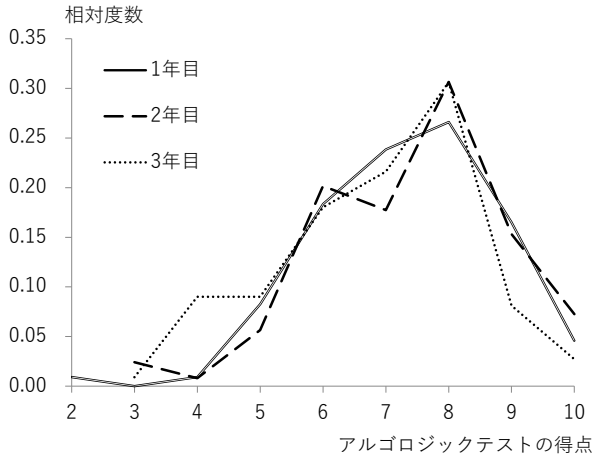


図 4 アルゴロジックテストのヒストグラム

図 4 に, アルゴロジックテストのヒストグラムを示す. 横軸はテストのスコア, 縦軸は相対度数である. アルゴロジックテストの問題数は 10 問, 1 問 1 点, 10 分が通常であり, 3 年目のみ 15 問, 1 問 1 点, 15 分であった. そのため, 図 4 では満点を 10 点として正規化している. 1 年目の平均点は 7.38 で標準偏差は 1.52, 2 年目は 7.31 で標準偏差は 1.44, 3 年目は 6.88 で標準偏差は 1.54 あり, Welch の t 検定で有意な差は見られなかった. 図 4 のとおり 3 群のヒストグラムも同様の形状であり, F 検定の結果統計的に有意な差は見られなかった. よって, 先に述べた平均 GPA での分析結果に加えて, 学力水準の分布もほぼ同様の集団とみなすことができる. なお, ここで重要な点として, 図 4 から得点が 6 割未満であった学習者が全体の 3 割程度存在していることが明らかとなった. アルゴロジックテストは, アルゴリズムを設計し構築することを要求せず, 命令を順に解釈し動作を想像するといった能力だけを求めている. 記述されたアルゴリズムを読解し適切に理解することは, プログラミング以前の基本的な能力であると考えられる. よって, この結果は, コーディングを行う前段階において, プログラミングの本質であるプログラミング的思考の前段階で求められる「形式言語の読解力」を十分に身に付けられていない学習者層の存在を示唆したものといえる.

アルゴロジックテストとプログラミングの各理解度テストの得点を図 5 に示す. 図 5 の箱ひげ図は, 25 パーセンタイルを第一四分位数, 50 パーセンタイルを第二四分位, 75 パーセンタイルを第三四分位数とし, 第一四分位数-1.5×IQR より下限の場合と第三四分位数+1.5×IQR より上限の場合を外れ値としている. 試験 1 と試験 2 は前期, 試験 3 は後期に実施されたものである. なお, 全てのテストの最大値は 10 で正規化されている. 図 5 から, 後半のテストほど難易度が高まるため得点は低下傾向にあり, 妥当な傾向が共通して確認された. これまでと同様に群間には統計的に有意な差は見られなかった.

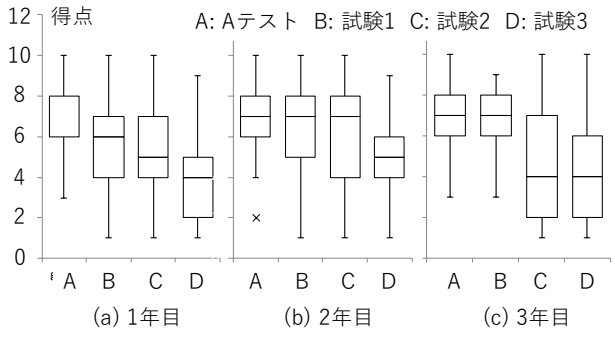


図 5 アルゴロジックテスト(A テストと表記)の得点分布と各理解度確認テストの得点分布(試験 1-3 と表記)

表 2 学習者数

	下位	上位
1 年目	49	55
2 年目	54	48
3 年目	59	44

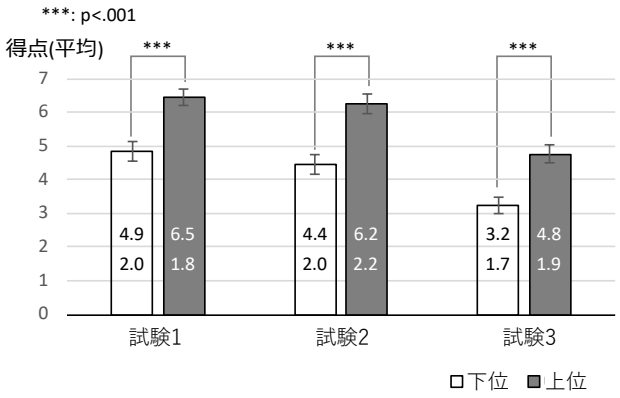


図 6 下位・上位群の各プログラミング理解度確認テストの平均値の比較(1 年目のデータ). 棒内に平均(上段)及び標準偏差(下段)を表記. 図 7, 図 8 も同様.

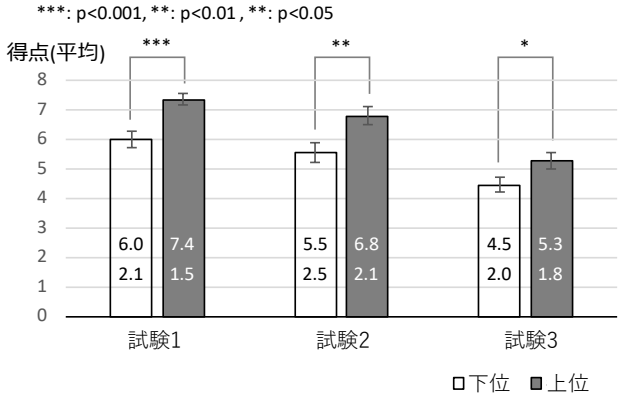


図 7 下位・上位群の各プログラミング理解度確認テストの平均値の比較(2 年目のデータ)

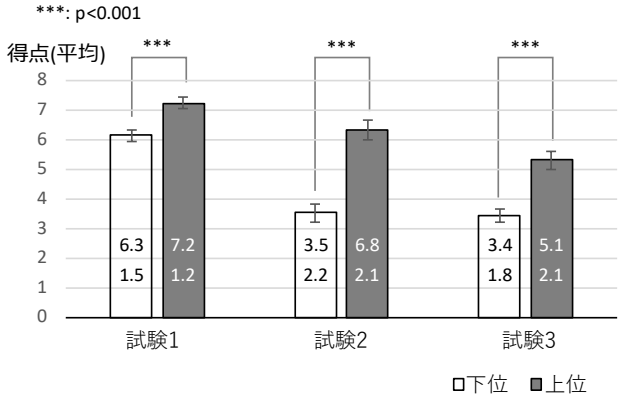


図 8 下位・上位群の各プログラミング理解度確認テストの平均値の比較(3 年目のデータ)

4. 2 アルゴロジックテストとプログラミング到達度との関係分析

アルゴロジックテストの得点と関連付けて, プログラミングを学習した後の到達度の差を調査した. 具体的には, アルゴロジックテストの得点に基づき全学習者を 2 群に分けて, プログラミングテストの平均値を群ごとに算出しその差を調査した. 本論文では, アルゴロジックテストで 8 割以上を獲得し「プログラミングの基本となる考え方をよく(とても良く, 完全に)できている」と評価された学習者群とそうでない群の 2 群に分割し, 前者を「上位グループ」, 後者を「下位グループ」とした. それぞれの母数については表 2 に示すとおりである. 下位学習者には増加, 上位学習者には減少の傾向がみられるが, カイ二乗検定の結果 p=0.41 となり有意な差は見られなかったため, 偶発的なことと仮定した. なお, 全てのプログラミングテストの得点は 10 段階で正規化したものを用いた.

結果を図 6, 図 7, 図 8 に示す. 縦軸はプログラミングテストの得点を表し, 試験ごとに学習者群の平均値を比較している. エラーバーは標準誤差を表す. 図 6, 図 7, 図 8 のとおり, すべての年で同様の傾向が示された. Welch の t 検定によると群間に統計的に有意な差が示され, 全ての下位グループの平均は上位グループの平均よりも低かった. このことからアルゴロジックテストパズルゲームとプログラミングテストとの間には正の関係が示唆された. 因果関係は明らかにできないため可能性の一つに過ぎないが, 正の関係が見られた理由として, いくつかの理由が考えられる. まず, プログラミングの知識を習得するためには順思考や読解など論理思考の前提となる何らかの適性が重要であり, その適性が不足しているためプログラミングの知識獲得が困難となっていた可能性である. プログラミング学習以前の初期段階においてプログラミングの基本的な考え方自体を受け入れられなかった学習者や, 当初より何らかの理由で(教授方法も含めて)そもそも学習自体に興味を持たなかった(拒絶反応を持った)学習者も同様であった可能性も考えられる. 基礎学力の影響を調査するため, 表 3 に示すように, 各学習者の総合 GPA(履修した科目全ての GPA の平均値)と各テストとの相関を算出した. なお, 相関は統計的に無相関ではなかった. 総合 GPA は基礎学力と仮定できるため, 表 3 の結果から「基礎学力が低い学習者

はアルゴロジックテスト(表内では A テストと表記)及びプログラミングテストも共に低い」といった可能性は小さい. アルゴロジックテストでは, アルゴリズム/命令の手順を適切に把握し, 出力を想像する力のみを確認するものである. 特定の入力に対する出力を考える必要はなく, さらに, アルゴリズムを自ら組み立てないため想像力も求めていない. アルゴロジックテストで確認している能力は, プログラミング言語の仕様を記憶してソースコードをコーディングするよりも基本的かつ重要な技能であり, プログラミングを学習するために前提である. このような観点で見ると, 今回の結果の背景には因果関係の存在も十分に考えられる. もちろん, 例外を無視できない. 具体的には, アルゴロジックテストの成績が良かった場合であっても実際のプログラミングテストが良くない学習者や, あるいはその逆の存在である. このような学習者の詳細を分析することは今後必要な課題であると考えられる. 例外の存在については表 4 に示すとおりである. 表 4 は 3 年間の全学生のデータから各テストの得点の相関行列を計算したものであり, 統計的に無相関ではない. 表 4 にみられるようにアルゴロジックテスト(表内では A テストと表記)とプログラミングテストの相関は十分に高いものではなく, 例外が多く含まれることを示唆している.

表 3 GPA と各試験の得点との相関行列

	A テスト	試験 1	試験 2	試験 3
GPA	0.12	0.27	0.21	0.29

表 4 GPA と各試験の得点との相関行列

	A テスト	試験 1	試験 2
試験 1	0.28		
試験 2	0.43	0.59	
試験 3	0.36	0.66	0.73

5. 議論

本論文の分析の結果, まず, 約 3 割の学習者は本格的なプログラミング学習を進めるために必要な考え方を学習以前に十分に身に付けられていなかったことを明らかにし, さらにこの傾向は 3 年間類似していたため一般傾向であることが示唆された. いくつかの先行研究では, プログラミング導入講義における不合格率に焦点を当てており, 世界規模での調査を実施している [26),27)]。この調査では, プログラミング言語, 学年, 国, 講義の規模に依存せず, 不合格率に実質的な違いはないことを明らかにしている. 一定の割合の学習者が毎年プログラミングの基本概念を理解していなかったことを明らかにした本論文の最初の分析結果は, 先行研究の知見を支持するものであり, 妥当な傾向が示されたものと考えられる. 次に, アルゴロジックテストの得点が 7 割以下の学習者のプログラミングテストの成績は, そうでない学習者よりも低いという傾向が示された. アルゴロジックテストでは, 命令の手順を適切に把握することのみを要求しているため, ここで確認している能力はプログラミングを学習するために前提となる不可欠な技能と言える. プログラミング初学者が経験した様々な問題は Robins らによって特定されているが [28)], 本論文はアルゴロジックというゲームを通じて初心者にとって最初の障壁を明確にできた点で意義があると言える. アルゴロジックテストの得点が低い学習者はまたプログラミングテストの成績が低いといった点について, ひとつの可能性として, 学習自体に興味がなかったのではないかと考えられる. Luxton-Reilly は, 学習初期に与えられる興味を持ちにくい難解な学習課題が事後のプログラミング学習を困難にすると述べている [29)]. 一方, 本論文ではそのような期待を持つ前段階でアルゴロジックを用いて比較的容易な演習を行い, この時点ですでに達成度の違いの存在を明らかにしている. この点で, プログラミング学習の困難性は初期の課題の構成に依存しない可能性が示されたといえる. Kinnunen が報告しているように [30)], 実験結果の背後にはモチベーションの欠如という要因が存在していた可能性も十分考えられる. そもそもアルゴリズムを考える・組み立てること自体に関心の低い学習者は, 学習初期にどのような教材を与えようともプログラミング学習を後に困難とする可能性が高いと考えられる. よって, 議論を発散させないためにも, 今後このような学習者を除外した分析が必要と考えられる.

プログラミングは大学などの高等教育機関の専門分野で特に重要な科目と見なされているため, プログラミングを顕著に理解できない学習者が全てのクラスに常に存在していることは一般的である. このような問題は二極化として広く知られている. 二極化について

は，改善可能とする楽観的な見解[31]と逆に改善は不可能とする悲観的な見解[32]がある．匂坂らは，理解度に応じて学習者をより厳密に分類し，プログラミングを特に困難とする学習者の特性を明らかにしている[33]．具体的には，プログラミングの理解度が最も低いグループに属する学習者のほとんどが，プログラミングの最も基本的な用語と文法を理解できていないことを確認している．さらに，そのひとつ上のグループのほとんどの学習者は，条件分岐，繰り返し，関数を全く理解できず，ソースコードを十分にトレースする技術も持たないことを明らかにしている．匂坂らの報告を考慮すると，最も低いグループのプログラミング学習者は，一般的なプログラムを書くよりも単純なプログラムを適切に読むための最低限の知識すら持っていないと考えられ，本論文で得た知見と一致するものと言える．

プログラミングが義務教育のカリキュラムに組み込まれた場合，プログラミングが苦手な学習者の存在は特に深刻な問題と考えらえる．そのため，いつどのようにプログラミング初学者が学習をあきらめる傾向にあるか理由を調査し，それを踏まえて新しい指導法を構築することは重要な課題と考えられる．本論文の取り組みは，学習前の段階で最下層グループとなる可能性が十分ある学習者を検知することに利用可能であり，二極化の改善に貢献できる可能性がある点で意義があると考えられる．

6. おわりに

本論文では，プログラミングゲームの一つとしてアルゴロジックに着目し，その理解度確認テストの結果から学習者の特徴を明らかにすると共に，アルゴロジックの理解度と実際のプログラミングの到達度との関係を調査した．本論文の分析の結果，まず，学習者の約 30%がプログラミングの本質的な学習を進めるために必要な十分な知識を持っていないことを明らかにし，さらにこの傾向は毎年同じであることを示した．次に，プログラミング学習前の事前状態とプログラミングを学習した後の事後状態の間に正の関係があることが明らかにされた．この分析結果は，アルゴリズム/データ構造を設計するための学習を行う前段階の技能が不十分であった可能性を示唆した．

本論文では大局的な観点からのみ分析結果を示している．したがって，様々な条件のもとで学習者を分類し学習者群ごとに傾向を分析するなど，本研究で示した結果の詳細を明らかにする必要がある．たとえば，本論文で例外的振る舞いを示した学習者の特徴や，逆にアルゴロジックテストの結果が顕著に悪かった学習者の特徴の詳細の分析などが考えられる．このような取り組みは，教授法や講義計画の設計，困難なアルゴリズム要素，講義に追従することが困難な学習者の抽出など，プログラミング教育をより充実させるために有用な知見が得られると考えられる[34]．

謝辞

本研究は，独立行政法人日本学術振興会科学研究費助成事業(基盤研究(C)17K01164，19K02987)による助成を受けて実施した成果の一部である．ここに記して謝意を表します．

【参考文献】

1) 野口孝文，千田和範，稲守栄．(2015)，初心者から上級者までシームレスにプログラミングを学ぶことができる持続可能な学習環境の構築．教育システム情報学会誌，32(1)，59-70.
2) Wing, J. M. (2006), Computational thinking. Communications of the ACM, 49(3), 33-35.
3) Aho, A. V. (2012), Computation and Computational Thinking. The Computer Journal, Vol.55, No.7, 832-835.
4) Bornat, R., Dehnadi, S. (2008), Mental models, consistency and programming aptitude. Proceedings of the tenth conference on Australasian computing education, Vol.78, 53-61.
5) Dehnadi, S., Bornat, R. (2006), The camel has two humps (working title). Middlesex University, UK, 1-21.
6) Bornat, R. (2014), Camels and humps: a retraction. URL http://www. eis. mdx. ac. uk/staffpages/r_bornat/papers/camel_hump_retraction.pdf, retrieved 2020/1/12.
7) Lahtinen, E., Ala-Mutka, K., Järvinen, H. M. (2005), A study of the difficulties of novice programmers. Acm Sigcse Bulletin, 37(3), 14-18.
8) Gomes, A., Correia, F. B., Abreu, P. H. (2016), Types of assessing student-programming

knowledge. In 2016 IEEE Frontiers in Education Conference (FIE), 1-8.
9) Gomes, A., Santos, Á. N. F. S., das Dores Páris, C. P., Martins, N. C. (2017), Playing with Programming: A Serious Game to Start Programming. In Gamification-Based E-Learning Strategies for Computer Programming Education, 261-277.
10) Parsons, D., Haden, P. (2006), Parson's programming puzzles: a fun and effective learning tool for first programming courses. In Proceedings of the 8th Australasian Conference on Computing Education, Vol. 52, 157-163.
11) Fal, M., Cagiltay, N. (2012), How scratch programming may enrich engineering education. In 2nd International Engineering Education Conference, 107-113.
12) Maloney, J., Burd, L., Kafai, Y., Rusk, N., Silverman, B., Resnick, M. (2004), Scratch: a sneak preview [education]. In Proceedings. Second International Conference on Creating, Connecting and Collaborating through Computing, 104-109.
13) Harvey, B., Mönig, J. (2010), Bringing “no ceiling” to scratch: Can one language serve kids and computer scientists. In Proceedings of Constructionism, 1-10.
14) Lewis, C. M. (2010), How programming environment shapes perception, learning and goals: logo vs. scratch. In Proceedings of the 41st ACM technical symposium on Computer science education, 346-350.
15) Ozoran, D., Cagiltay, N., Topalli, D. (2012), Using scratch in introduction to programming course for engineering students. In 2nd International Engineering Education Conference, Vol.2, 125-132.
16) 佐賀孝博. (2012), アルゴロジックとプログラミンを用いたプログラミング学習. 稚内北星学園大学紀要, 12, 99-111.
17) Cooper, S., Dann, W., Pausch, R. (2003), Teaching objects-first in introductory computer science. In ACM SIGCSE Bulletin, Vol.35, No.1, 191-195.
18) Cheung, J. C., Ngai, G., Chan, S. C., Lau, W. W. (2009), Filling the gap in programming instruction: a text-enhanced graphical programming environment for junior high students. ACM SIGCSE Bulletin, 41(1), 276-280.
19) Pasternak, E. (2009), Visual Programming Pedagogies and Integrating Current Visual Programming Language Features. Master's Thesis, Tech. Report, Robotics Institute, Carnegie Mellon University, August, 2009
20) Warth, A., Yamamiya, T., Ohshima, Y., Wallace, S. (2008), Toward a more scalable end-user scripting language. In Sixth International Conference on Creating, Connecting and Collaborating through Computing, 172-178.
21) Fraser, N. (2013), Blockly: A visual programming editor. URL: https://code. google. com/p/blockly, retrieved 2020/1/12.
22) Ingalls, D., Kaehler, T., Maloney, J., Wallace, S., Kay, A. (1997), Back to the future: the story of Squeak, a practical Smalltalk written in itself. In ACM SIGPLAN Notices, Vol.32, No.10, 318-326.
23) Matsuzawa, Y., Tanaka, Y., Sakai, S. (2016), Measuring an impact of block-based language in introductory programming. In International Conference on Stakeholders and Information Technology in Education, 16-25.
24) Nishida, T., Harada, A., Yoshida, T., Nakamura, et al., (2008), PEN: A Programming Environment for Novices—Overview and Practical Lessons—. In EdMedia+ Innovate Learning, 4755-4760.
25) Roque, R. V. (2007), OpenBlocks: an extendable framework for graphical block programming systems (Doctoral dissertation, Massachusetts Institute of Technolo-

gy).
26) Bennedsen, J., Caspersen, M. E. (2007), Failure rates in introductory programming. AcM SIGcSE Bulletin, 39(2), 32-36.
27) Watson, C., Li, F. W. (2014), Failure rates in introductory programming revisited. In Proceedings of the 2014 conference on Innovation technology in computer science education, 39-44.
28) Robins, A., Rountree, J., Rountree, N. (2003), Learning and teaching programming: A review and discussion. Computer science education, 13(2), 137-172.
29) Luxton-Reilly, A. (2016), Learning to program is easy. In Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education, 284-289.
30) Kinnunen, P., Malmi, L. (2006), Why students drop out CS1 course?, In Proceedings of the second international workshop on Computing education research, 97-108. ACM.
31) Höök, L. J., Eckerdal, A. (2015), On the bimodality in an introductory programming course: an analysis of student performance factors. In 2015 International Conference on Learning and Teaching in Computing and Engineering, 79-86.
32) Basnet, R. B., Payne, L. K., Doleck, T., Lemay, D. J., Bazelais, P. (2018), Exploring Bimodality in Introductory Computer Science Performance Distributions. EURASIA Journal of Mathematics, Science and Technology Education, 14, 10.
33) 匂坂智子, 渡辺成良. (2010), プログラミング初学者のための Web–based 学習診断システムの開発と評価. 教育システム情報学会誌, 27(1), 29-38.
34) Brennan, K., Resnick, M. (2012), New frameworks for studying and assessing the development of computational thinking. In Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada, Vol.1, 25.

受付: 2019 年 11 月 20 日
採録: 2020 年 2 月 22 日

――― 著者略歴 ―――

松本 慎平　2007 年大阪大学大学院博士後期課程修了. 2010 年広島工業大学情報学部助教, 2013 年より同大学准教授となり現在に至る. 教育工学, 社会システム工学の研究に従事. 本学会の他, 電気学会, 教育システム情報学会などの各会員. 博士(情報科学). 本研究では, プログラミング教育の実践・原稿執筆・編集委員会との対応を担当.
E-mail: s.matsumoto.gk@cc.it-hiroshima.ac.jp

加島 智子　2010 年大阪大学大学院情報科学研究科博士後期課程修了. 同年近畿大学工学部助教を経て, 12 年近畿大学工学部講師, 現在に至る. 農業情報システム, 教育評価手法などに関する研究に従事. 本学会の他, 教育システム情報学会等の各会員. 博士(情報科学). 本研究では, 評価手法の構築, 評価結果の分析を担当.
E-mail: kashima@hiro.kindai.ac.jp

山岸 秀一　1984 年東京大学理学部物理学科卒業. 同年(株)三菱総合研究所入社. 産業技術総合研究所, 三菱電機(株)を経て, 2011 年より広島工業大学情報学部教授. 物理シミュレーション, 映像信号の符号化方式, 画像処理, 教育工学の研究に従事. 博士(理学). 本研究では, プログラミング教育の実践を担当.
E-mail: s.yamagishi.if@it-hiroshima.ac.jp