情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2024-CN-121 No.48
Vol.2024-CDS-39 No.48
Vol.2024-DCC-36 No.48
2024/1/23

Repro4PDE: コード編集後に動作と操作を任意の地点に復元可能な開発システムの実現

古賀壮[1]　中村聡史[1]

概要：プログラムを作成する際，コードを書く・実行する・修正するというサイクルを頻繁に回すことは重要である．ここで，コードの修正後に確認で再実行をするとき，プログラムは最初から実行される．通常はこうした挙動で問題ないが，実行に時間が掛かるアニメーションプログラムや，マウスやキーボード操作が必要なインタラクティブなプログラムの場合，元のコードでバグが起きていた位置まで進むのを待つこと，マウスやキーボード操作を行って問題ないかを確認することは手間になる．そこで本研究では，コード編集後の再実行時に修正前の実行状態をユーザの操作も込みで復元できる Processing 向けのツール Repro4PDE を実装した．その結果，アニメーションプログラムのパラメータ調整を行うというタスクにおいて試行回数の増加や所要時間の減少が見られたものの，その動作の把握の難しさから正解率は向上しないという問題もあり，今後の課題が明らかになった．

キーワード：プログラミング支援，Processing，アニメーション，パラメタ調整，効率化，再実行

1. はじめに

プログラムを作成する際，コードを書く・実行する・修正するというサイクルを頻繁に回すことは重要である．ここで，実行において生じたバグを直すために，コードを修正して再実行をするとき，コードは最初から実行される．通常はこうした挙動で問題ないが，実行に時間が掛かるアニメーションプログラムの途中に問題がある場合や，マウスやキーボード操作が必要なインタラクティブなプログラムで，その操作後にバグが発生するような場合，バグが起きていた位置まで進むのを待ったり，マウスやキーボード操作を行って問題ないかを確認したりすることは手間になる．

ここで，コードをすべて再実行するのではなく部分的に修正および実行可能な仕組みに，Jupyter Notebook などで実現されているステップに分解した逐次実行や，部分修正とその修正に応じた再実行などの仕組みがある．他にも，部分的にリアルタイムに実行結果を変化させることが可能な言語にライブプログラミング言語があり，実行しながらの即時のフィードバックが可能である．また，Web アプリケーションではホットリロードのような即座に実行状況を確認可能な仕組みもあり，開発の効率化が可能である．こうした仕組みは，先述のような毎フレームユーザが定義した描画関数が呼び出されるアニメーションプログラムや，様々な操作を再現して確認するようなインタラクティブなプログラムの問題を解決することはできない．

ここでアニメーションやインタラクティブなプログラムにおいて重要なのはその実行状況および操作に基づく結果の再現と，その問題が発生した部分から少し巻き戻して再実行することである．例えば実行に時間が掛かるプログラムとして，図 1 で示すラングトンの蟻がある．これは適当な初期状態を与えると初めはでたらめに見える動きをするが，約 10000 ステップ（200 秒）後に「道」と呼ばれる画面外に蟻が走っていくようなパターンを作るセルオートマトンである．このプログラムにおいて「道」ができたとき，配列の範囲外チェックを忘れたことが原因でエラーが発生することがある．実際に，この課題は明治大学総合数理学部先端メディアサイエンス学科のプログラミング演習の講義で用いられているが，多くの学生が範囲外チェックを忘れたことで実行時エラーが発生するというバグに遭遇している．このエラーの原因と思われる箇所を修正し，再実行するとき，動作確認のために約 10000 ステップ（200 秒）待つ必要があり手間である．

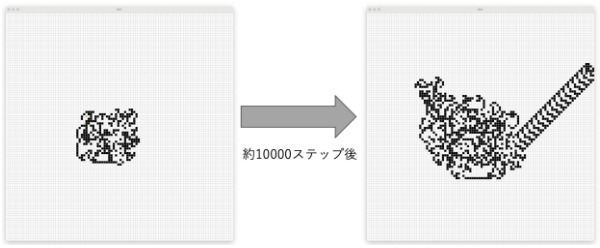

図 1 ラングトンの蟻

そこで本研究では，このような背景を踏まえ，プログラムの動作確認の手間を減らし，開発の効率化を支援することを目的とし，アニメーションプログラムを作る言語である Processing を対象に再実行の手間を減らすツール Repro4PDE を提案し，システムとして実装する．また，実験によりシステムの特徴について明らかにする．

2. 関連研究

2.1 頻繁にフィードバックを得られるシステム

頻繁にプログラムの実行結果というフィードバックを得られるシステムの研究がいくつかある．Allen ら [1]は REPL を用いた即時のフィードバックができる教育用 Java 開発環境を提案している．又吉ら [2]は 1 行書くごとにプログラムが逐次実行されるタイピングシステムを提案し，

Vol.2024-CN-121 No.48
Vol.2024-CDS-39 No.48
Vol.2024-DCC-36 No.48
2024/1/23

授業前の課題として使った結果プログラミング習得に効果があったことを明らかにしている．Singer ら [3]は Haskell のインタラクティブなコード実行環境を用いたオンラインコースを実施した．

2.2 プログラミング学習用言語としての Processing

本研究で対象とする Processing や，JavaScript 版である p5.js はプログラミング初学者向きの言語である．Terroso ら [3]は非プログラマ向けにクリエイティブなプログラムを p5.js で作る授業を行い, 離脱者が少なかったことと好意的なフィードバックを得られたことを報告した．Greenberg ら [4]は Java より Processing の授業のほうが追加でコンピュータサイエンスの授業を履修することや，課題に自主的に多くの時間を費やすといった行動が見られることを報告している．Bälter ら [5]は Python・Processing・Java を使った授業を行った．このような理由から提案システムは初学者の教育支援にもなる．

2.3 ライブプログラミング

即時的なフィードバックが得られるプログラミング環境のひとつにライブプログラミングがある. Kubelka ら [6]はプログラマがライブプログラミング言語の特性を，動作結果の仮説を確かめることなどに使っていることを明らかにしている．Tillmann ら [7]は UI を作るライブプログラミング言語を開発し, Burckhardt [8]らはその定式化を行った．BeckmannTom ら [9]はゲーム開発におけるライブプログラミング環境を提案した．その結果イテレーションが短くなり，デバッグの質が向上したことを報告している．

2.4 インタラクティブにコードを書く環境

また，動作確認をしながらコードを書くシステムとしては，NørmarkKurt ら [10] は Lisp を対象とした REPL を用いたインタラクティブなテスト作成環境を提案し，テストの作成が容易になることにより，ソフトウェアの品質が向上する可能性を示した．Lahiri ら [12]は LLM を用いてインタラクティブにテストを書く手法を提案した．

3. 提案システム

3.1 必要要件

プログラムのバグを修正している際に，その動作確認のため動作確認したい位置まで進めたいという状況は頻繁にある．ここで，その位置までプログラムを進めることが容易ではない状況として，まず 1 章で紹介したラングトンの蟻のような課題が存在する．この課題の場合，コードに含まれる問題を発見し，コードを部分的に修正しただけなのに，正常に実行できていることを確認するためはまた 200 秒待たなければならず手間である．また，そこでさらに修正したコードに間違いがあった場合などは，また追加で 200 秒待つ必要があるなど，時間を無駄にしてしまうものである．

また, 4.2 節の実験において後述するコンプリートガチャのプログラムは，クリックのたびにランダムに値を発生させ，得られた値に基づきカードの数を増やし，カードの枚数がすべて 1 枚以上になったら全てのカードを赤くするというものである．ここで,「すべて 1 枚以上になった」という判定がうまくいかずにコードを修正した際，また最初からクリックを繰り返していくという手間がある．こうした状況において，これまで行ったクリック操作を再現し，そのすべて 1 枚以上になったという判定をおこなう状況まで瞬時に移動できることが望ましい．なお，乱数を用いて値を取得する場合は，同じシード値を用い，同じ数値を順に取得できることが確認において望ましい．

また，動的に変化するプログラムを実行している際に，その問題のあるポイントが一瞬で過ぎ去ってしまい，その確認が困難になってしまうという問題がある．こうした際には，また最初から再度実行しなければならなくなる．デバッガなどを用い，順にステップ実行して確認することは可能であるが，アニメーションプログラム等の場合，かなりのステップ数を飛ばす必要があり，その必要な場所を確認するのは容易ではない．このような時に，進みすぎたプログラムを，指定したステップ数分手軽に戻すことができればその確認が容易になる．

一方，プログラムを修正する際に元のプログラムからどこを書き換えたのか，元のプログラムの動作はどうなっていたのかを記憶しておくことも容易ではない．元のプログラムをバックアップしておくことで比較することも可能ではあるが，バックアップ自体の手間に加えてインタラクティブなプログラムであれば両方のプログラムに対して同じ操作をしなければいけないという課題がある．

以上の要件をまとめると以下のようになる．

- プログラムの再実行時に, プログラムを前回と同じ状態にすることを容易にする
- 実行位置を好きな位置に巻き戻すことができる
- コード変更時にプログラムのどこを変更した結果どのように動作が変わったのかを一目で確認する

そこで本研究では，これらの要件を満たすシステム Repro4PDE を提案および実装する．

3.2 提案手法

プログラムの再実行時に前回と同じ状態にすることを容易にするため，まずその実行状態までのステップ数を記憶しておき，プログラムにおける無駄な描画処理や，内部的な待機時間などをスキップすることで，元の状態への再現を容易化する状態復元手法を実現する（詳細は 3.3 節において後述）. また, 同一の状態を再現することを重視するため，乱数の生成を行っている場面では，同じシード値を使うことで，ランダム関数を用いていながら同様の状況を再現する．さらに，ユーザのマウスやキーボードによる操作も記録しておき，その操作を自動実行することにより，

Vol.2024-CN-121 No.48
Vol.2024-CDS-39 No.48
Vol.2024-DCC-36 No.48
2024/1/23

ユーザの操作も考慮した実行状態の復元を可能とする．

また，上述の仕組みを利用することによって，実行位置の任意の場所へのスキップを容易化するとともに，メディアプレイヤのようなシークバー型のインタフェースを用意し，そのシークバーの操作により，任意の実行位置までジャンプし，実行状態を復元可能とする．なお，シークバー型のインタフェースだけがあっても，どの部分に戻したら良いかわからないと考えられる．そこで，シークバー上にマウスカーソルをインタフェース，その実行状態を確認可能とする．

また，プログラム修正後の比較を容易にするためには，プログラム再実行時に，コードの差分と，プログラムの動作の差分を同時に提示することが理解を促すと考えられる．そこで，そうした比較機能を提供することで解決する．

3.3 状態復元手法

実行状態の復元のために，実行開始からのフレーム数・各フレームで発生したイベント列・乱数のシード値を記録しておき，特定のフレームまでイベントや乱数のシード値を再現しながら実行し，状態を復元する．この時，状態復元を高速に行うため，1 フレームの間に行う処理のうち，変数の値の更新など図 2 の青色で示す必要な処理のみを行い，描画処理や，1 フレームの全ての処理が指定のフレームあたりの秒数に満たなかった場合に Processing 内部で行われる待機処理などの，黄色で示す不要な処理を省略するようにした．

3.4 実装とシステム概要

本システムは対象とする Processing と同じく Java Virtual Machine（JVM）上で動く言語である Scala で実装し，UI には JavaFX を利用した．また，Processing のツール（プラグイン）として実装されているため，Processing のメニューから起動することができ，起動すると図 3 で示すウィンドウが表示され，各種機能を利用することができる．

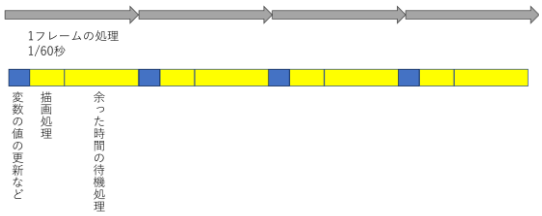

図 2 早送りによる状態復元

図 3 システムの UI

システムのアーキテクチャを図 4 に示す．各種機能を実現するためには実行対象のプログラムに対して様々な変更を加える必要がある．そこでビルド時のコード書き換えと実行時のランタイムの挿入という 2 つの方法によって実現した．

まず，システムの再生ボタンが押されるなどして実行が開始された時，Processing のビルド機能を使ってプログラムをビルドするが，この時 Java の Reflection 機能を使うことでビルド処理に介入し，コードを書き換える．これは Processing のライブラリを書き換えるのに使っており，例えば描画が必要ないタイミングでは各種図形描画関数の処理をスキップするといったライブラリの書き換えを行った．

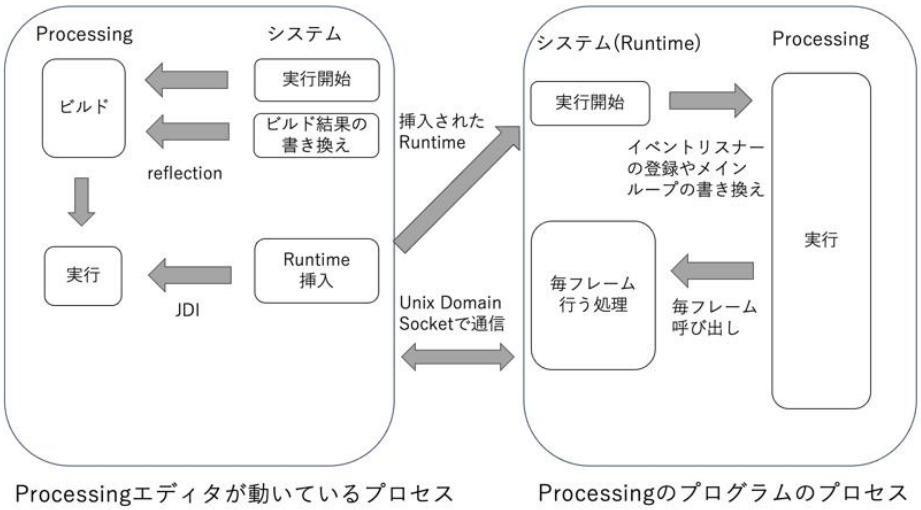

図 4 システムのアーキテクチャ

Vol.2024-CN-121 No.48
Vol.2024-CDS-39 No.48
Vol.2024-DCC-36 No.48
2024/1/23

また，実行が開始された後は Java Debug Interface（JDI）によって独自のランタイムを挿入している．ランタイムの初期化処理ではメインループの書き換えや，Processing に毎フレームランタイムの関数を呼び出させるためにイベントリスナーへの登録などを行っている．挿入させたランタイムは発生したイベントを記録したものや，現在の実行位置などを Unix Domain Socket を使って Processing エディタと同じプロセスで動いているシステムに送信したり，反対にシステムから一時停止コマンドなどを受け取ったりすることで連携している．

3.5 機能

システムはメディアプレイヤのような UI をもっており，再生・一時停止・停止・再生位置の変更といった操作を行なったり，シークバーの上にマウスをホバーすることで，特定の再生位置の画面プレビューをポップアップで確認したりすることができる．また，ユーザが再生ボタンを押すとランタイムが挿入され実行される．その後，ユーザが停止ボタンを押してプログラムを変更し，再生ボタンを再度押すとプログラムは初めから実行されるのではなく，状態復元が行われ停止時の再生位置から再生が行われる．なお，ユーザがボタン操作を行わなくても，再生中にプログラムを変更し，保存をすると自動で再起動し，再実行することもできる．

また，ユーザはシークバーを利用することで，指定の位置まで実行状態を戻すことが可能である．内部的には，シークバーの位置が変更されると，プログラムを再起動し，指定された再生位置まで状態を復元することで実現している．

これらの機能を用いてバグのあるラングトンの蟻のプログラムを修正する場合， 図 5 のようにプログラムを修正すると約 10000 ステップ分を数秒で実行し，エラーが発生しなくなったことを確認できる．

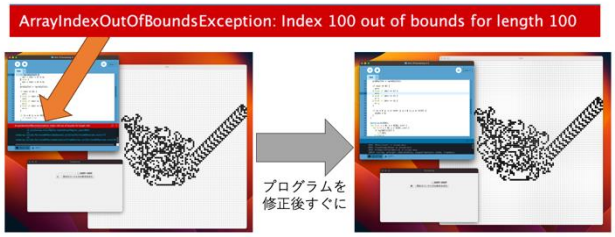

図 5 バグの修正例

また，修正前後のプログラムの比較を容易にするための機能として「現在のコードとの比較」機能がある．これはソースコード上の差分と，実行結果の差分を一目で確認できる機能である．2 つの実行結果間では再生位置や発生したイベントが同期され，ソースコードの差分も表示される．この機能を使って，インタラクティブなプログラムの一種であるお絵描きプログラムにおける描画色と線の太さの変更を行った様子が図 6 である．この図のようにどのようなソースコードの修正によってどう動作が変わったかを一目で確認することができる．

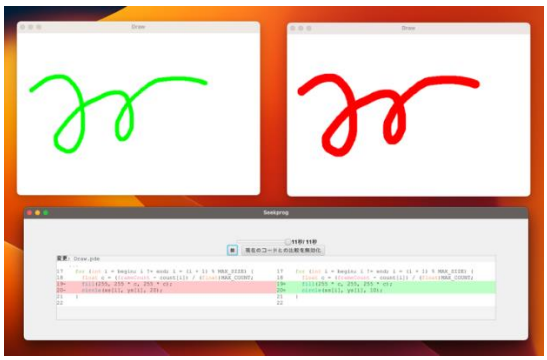

図 6 現在のコードと比較モード

4. 実験

本実験では，提案システムを使うことで動作確認の手間が減少し，プログラムの開発効率が上がるのかについて調査する．

4.1 実験概要

実験では，提案システムを使い，指定したプログラムを完成させる練習用タスク 1 つと，実験用タスク 3 つを解くことを依頼した．問題は pdf で配布し，完成例として著者が作ったプログラムのスクリーンショットを添付した．参加者は同じ研究室に所属する 10 人であり，全員学部 1 年生の時に，学科の必修授業で類似の課題を解いた経験がある．ここでは，再実行時の状態復元ありグループと，再実行時の状態復元なしグループに 5 人ずつに分けて比較する実験を行った．

実験は，使い慣れた個人仕様の PC に事前に，Processing と Repro4PDE をインストールしてきてもらい行った．ここではまず，Repro4PDE システムの使い方を説明しながら練習用タスクを 20 分，本番用タスク 3 つを 10 分，20 分，40 分で取り組んでもらった．また，終了後に問題の難易度などに関するアンケートを行った．

4.2 実験用のタスク

練習用タスクでは，マウスでクリックした位置にランダムな色で円を描画するプログラムを作ることを課した．このタスクを行なってもらいながら使い方を説明することで状態復元の動作や，乱数のシード値の固定の意味を理解してもらった．

本番用タスクの 1 つ目（Q1）はパラメータ調整タスクである．このタスクでは，初速度と摩擦係数が適当に設定されており，壁に複数回跳ね返ったあと一定速度を下回ると球が停止するプログラムを配布し，そのプログラムを利用して 5 回以上壁にぶつかったあと中央で球が停止するよう

Vol.2024-CN-121 No.48
Vol.2024-CDS-39 No.48
Vol.2024-DCC-36 No.48
2024/1/23

に初速度と摩擦係数の値を調整することを課した．このタスクは実行に時間が掛かるアニメーションプログラムにおいて，細かいパラメータ調整の支援になるかを確認するために設定した．このプログラムのパラメータを適切に調整すると図 7 のような画面で停止する．

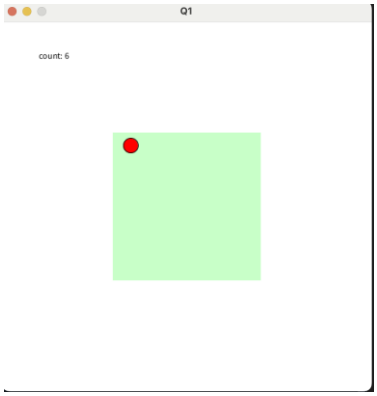

図 7 パラメータ調整タスクの調整例

本番用タスクの 2 つ目（Q2）は，コンプリートガチャタスクである（図 8）．これは，ボタンをクリックすると 5 枚のカードから 1 枚をランダムで獲得でき，獲得した枚数をそれぞれ表示する．また全てのカードを 1 枚以上手に入れたら，コンプリートとして全てのカードを赤くすることを課した．このタスクは，乱数が絡み，またユーザ操作が必要なインタラクティブなアプリケーションにおいて，本システムが効果的に働くかを確認するために設定した．

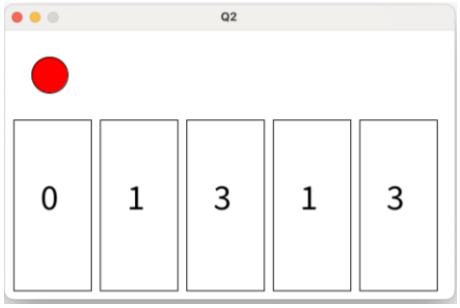

図 8 コンプリートガチャタスクの完成例

本番用タスクの 3 つ目（Q3）はマインスイーパタスクである．このタスクでは，事前にランダムに爆弾を 1 つ隠して配置し，ユーザがクリックした隣接マスに爆弾があるかを調べ，クリックしたマスの隣接マスに爆弾がない場合はその隣接マスをすべて開放し，隣接マスに爆弾が含まれる場合はそのマスのみを開放，爆弾をクリックしてしまったら画面を赤で塗りつぶすというプログラムを作るというものである．このタスクも，乱数が絡み，またユーザの操作が必要なインタラクティブなアプリケーションで，かつ周りの爆弾の個数を確認するなど様々な判定があるなどの状況において効果的に働くかを確認するために設定した．

4.3 実験用システム

システムは実験のためにいくつかの調整を行なった．まず，分析のために再生・停止といった各種操作ログをタイムスタンプと操作時のソースコードを含めてログとして記録するようにした．また，誤操作を防ぐためにシステムを簡単に終了できないようにし，Processing のエディタについている再生・停止ボタンを無効化した．

今回，システムを使い慣れていないユーザを対象に実験を行うため UI をシンプルにし，また状態復元のありなしによる差のみを比較するために，「現在のコードとの比較」機能と，保存時に自動で再起動する機能を無効化した．実験で使ったシステムの UI を図 9，10 に示す．

図 9 実験用システム（状態復元あり群）

図 10 実験用システム（状態復元なし群）

5. 実験結果

状態復元ありグループ（A）となしグループ（B）で比較した問題ごとの正答数を表 1 に示す．

表 1 正答数

	状態復元あり(A)	状態復元なし(B)
Q1	3	3
Q2	2	4
Q3	1	5

表 1 より，全問正解者は A グループに 1 人，B グループに 3 人おり，提案手法を使った A グループの方が，正当数が少ないことがわかる．

回答時間の箱ひげ図を図 に示す．なお平均回答時間は

Vol.2024-CN-121 No.48
Vol.2024-CDS-39 No.48
Vol.2024-DCC-36 No.48
2024/1/23

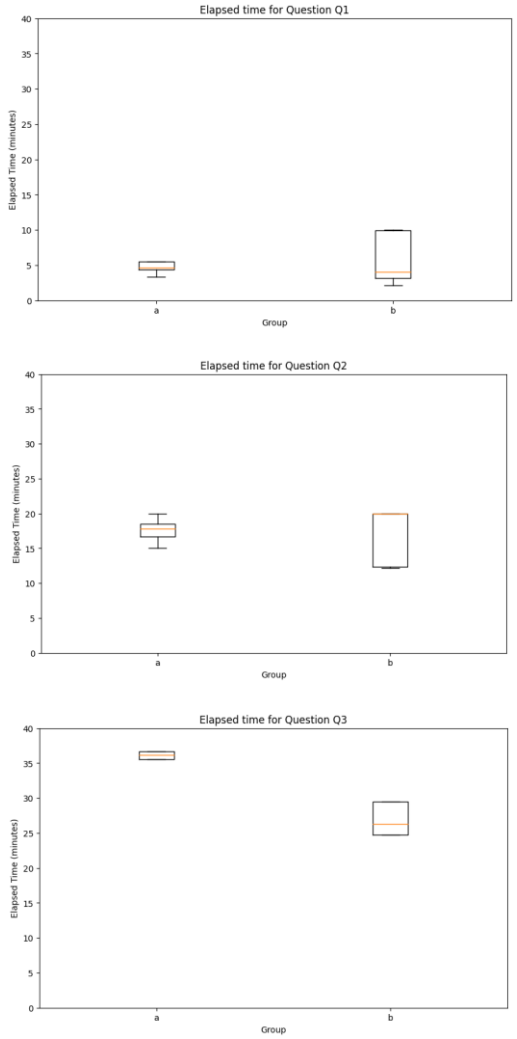

図 11 回答時間の箱ひげ図（上から Q1，Q2，Q3）

時間内に終わらず不正解となった参加者はタスクの制限時間で計算している．この結果より，システムを使った群のほうが分散が小さく，Q1 についてはほとんどの人が短時間でタスクを終わらせていることが分かる．一方，Q3 については，達成できなかった実験協力者も多かったことから，かなり悪い結果となっていることがわかる．動作確認を手軽に行えるようになったかを調べるために参加者がいつ実行ボタンを押したかのデータを分析した．平均実行回数のグラフを図 12 に，タスク開始からの時間を横軸とした実行回数の分布を図 13 に示す．この結果より，Q1 と Q3 で実行回数が増えていることがわかる．

実験後，参加者にアンケートを行った．アンケートの結果より，復元機能は多くの人が Q1 で利用しており，Q2 と Q3 については活用できるまで完成しなかったと答えた人が多かった．ただし，一部の人は Q2 と Q3 のクリックした後の画面がどうなるかについて確認するのに使ったと答えていた．

またシステムの改善点について，Q2 と Q3 についてコー

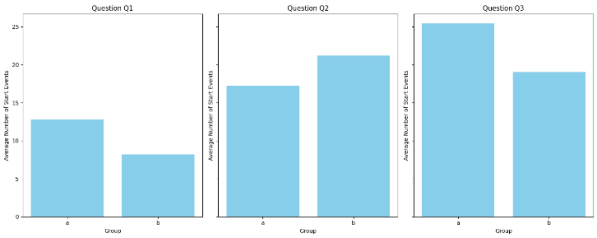

図 12 平均回答時間

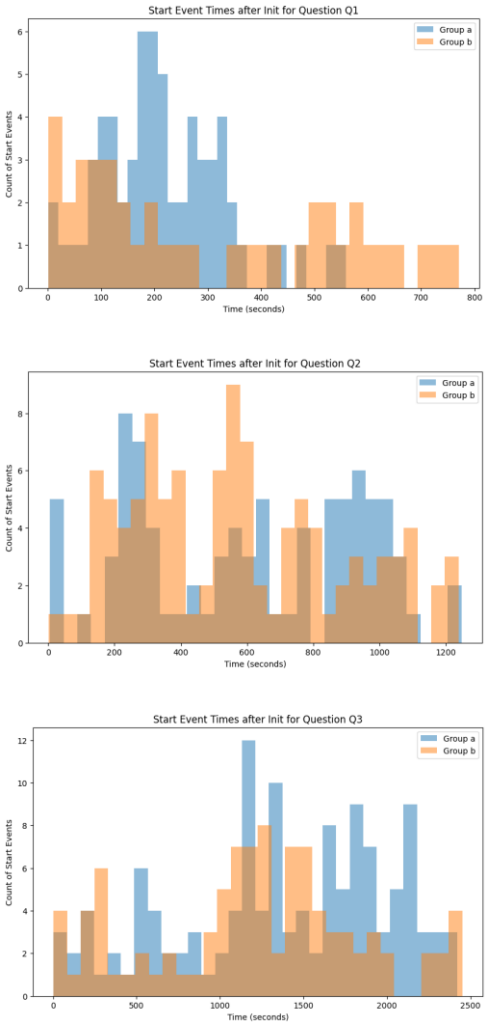

図 13 実行回数の分布

ドを修正し，状態復元をした後に画面が異常な状態になった際に，どこでバグが起きたのか分かりにくいことや，余計なマウスクリックまで座標で記録されていることによる混乱などが挙げられた．また，変数の値を確認する機能が欲しいという回答もあった．

6. 考察

6.1 タスクごとの正答率と回答時間に関する考察

正答率が同じ Q1 については，A グループのほうが回答時間の分散が小さい．また，タスク開始直後に多くの実行

Vol.2024-CN-121 No.48
Vol.2024-CDS-39 No.48
Vol.2024-DCC-36 No.48
2024/1/23

を短時間に行っていることから，このようなタスクについては状態復元によって動作確認が効率的になったのではないかと考えられる．

正答率は B グループのほうが Q1 を除いて高かった．このような結果になった理由として，Q2 と Q3 のロジックの複雑さがあげられる．提案システムは画面が時間経過でどう変化していくのかといったアニメーションや，画面をクリックしたらどうなるのかといったインタラクティブ性のあるプログラムの作成をサポートするシステムであるが，Q2 の「全てのカードを手に入れた時」や Q3 の「クリックした周囲のマスに爆弾があるか」を判定するロジックの作成や，状態をどう持つのかといった設計をサポートするシステムではない．アンケートでこのような部分の実装が難しかったことや，時間が足りなかったことをあげている人が多いことから，提案システムが得意な結果画面がどうなるかといった調整までたどり着かなかったのではないかと考えられる．また参加者は全員提案システムに初めて触れる人であったため，慣れていないシステムを使うことによるデメリットが，システムのサポートによる利点を上回ったのではないかと考えられる．今後は，こうした問題を解決し，長期的な利用からの有用性を検証予定である．

6.2 システムの改善

アンケートではロジックの作成が難しかったということに加え，変数の値を見ることができれば便利という意見があった．提案システムは状態設計やロジックの作成を支援するものではないが，既存のデバッガと共存して使えるようにするといった改善は必要であると考えられる．

また，実験の結果インタラクティブなプログラムにおいて提案システムを使う難易度が高いことが分かった．提案システムはクリック操作履歴全てを座標ベースで記録するため，ユーザの意図しない操作の再現が行われた可能性がある．例えば，イベント処理を実装していない状態で無意識に画面をクリックし，クリックに関する処理を実装した後に再起動すると，無意識なクリックによる処理が実行された後の状態が復元される．このため，再実行後になぜ表示された画面になったのか分からないという問題が発生した可能性がある．この問題を解決するには，システムに慣れてもらう必要があると考えられ，長期的な利用をしてもらうことで検証する必要がある．また，状態復元後にどの操作が再現されたのかということを分かりやすく提示するといった改善が必要である．

実際にアンケートでも状態復元後になぜ提示された画面になったのか分からなかったといった意見や，実験中の操作を観察した結果, Q1 では状態復元された画面で動作確認をしている人が多かったのに対して，Q2 と Q3 では再起動後にシークバーを 0 に戻し，最初から実行している人が多いことが分かった．

7. まとめ

本研究では，実行に時間が掛かるアニメーションプログラムや，マウスやキーボード操作が必要なインタラクティブなプログラムにおいて，コードの修正後にプログラムは最初から実行されることによる待機時間と手間に注目し，プログラムの修正から再実行における動作確認を容易にするシステムを提案しシステムとして実装した．また 3 つのタスクを用いた実験を行い，結果アニメーションプログラムのパラメータ調整タスクでは試行回数の増加や所要時間の減少が見られたが，インタラクティブな乱数の絡むプログラムを作る課題では正答率が減少するなどの問題が確認された．

今後は，実験の結果から提案システムを使うのに向いているプログラムの種類や，タスクを解く上で課題となっている問題点が明らかになったため，そうした問題を踏まえた改善を行っていく予定である．また，慣れにより改善できる点もあるため，長期的な利用による検証を行っていく予定である．

参考文献

[1] E. Allen, R. Cartwright , B. Stoler, “DrJava: A lightweight pedagogic environment for Java,” 著: *Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, 2002.

[2] 又吉康綱 , 中村聡史, “typing. run: 初学者のプログラミング学習を支援するプログラムタイピングシステムの提案と実践,” 2020.

[3] T. Terroso , M. Pinto, “Programming for Non-Programmers: An Approach Using Creative Coding in Higher Education,” 著: *Third International Computer Programming Education Conference (ICPEC 2022)*, Dagstuhl, 2022.

[4] I. Greenberg, D. Kumar , D. Xu, “Creative coding and visual portfolios for CS1,” 著: *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, 2012.

[5] O. Bälter , D. A. Bailey, “Enjoying Python, processing, and Java in CS1,” *ACM Inroads,* 第 巻 1, p. 28‑32, 2010.

[6] J. Kubelka, R. Robbes , A. Bergel, “The road to live programming: insights from the practice,” 著: *Proceedings of the 40th International Conference on Software Engineering*, 2018.

[7] S. Burckhardt, M. Fahndrich, P. De Halleux, S. McDirmid, M. Moskal, N. Tillmann , J. Kato, “It's alive! continuous feedback in UI programming,” 著: *Proceedings of the*

Vol.2024-CN-121 No.48
Vol.2024-CDS-39 No.48
Vol.2024-DCC-36 No.48
2024/1/23

34th ACM SIGPLAN conference on Programming language design and implementation, 2013.

[8] N. Tillmann, M. Moskal, J. De Halleux , M. Fahndrich, “Touchdevelop: Programming cloud-connected mobile devices via touchscreen,” 著: *Proceedings of the 10th SIGPLAN symposium on New ideas, new paradigms, and reflections on programming and software*, 2011.

[9] T. Beckmann, E. Krebs, P. Rein, S. Ramson , R. Hirschfeld, “ Shortening Feedback Loops in a Live Game Development Environment,” 著: *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2021.

[10] K. Nørmark, “Systematic Unit Testing in a Read-eval-print Loop.,” *J. Univers. Comput. Sci.,* 第 巻 16, p. 296‐314, 2010.