時間とのインタラクションによるプログラミング支援

加藤邦拓[†]　　宮下芳明[†,††]

本稿では，時間とのインタラクションによるプログラミング支援システムを提案する．本システムは時間軸を超えて実行可能な「実行画面表示ウィンドウ」と，それに連動して時間を移動できる「ソースコードエディタウィンドウ」を持つ．これらはコードの入力が行われる度に自動でプログラムを実行し，その結果を表示する．これによりソースコードの作成を行いながらリアルタイムにプログラムエラーの有無を知ることができる．また，パラレルに存在する任意の過去に自由に移動することが可能となり，より柔軟に時間軸を超えた作業を進めることができる．

Programing Support Based on Interaction with The Time

KUNIHIRO KATO[†]　HOMEI MIYASHITA[†,††]

In this paper, we propose a programing support system by the interaction with the time. This system has the “execution display window” which can be performed exceeding a time-axis, and the “source code editor window” which is interlocked with it and can move time. Thereby, the existence of a program error can be known in real time, creating a source code. Moreover, it becomes possible to move to the arbitrary past which exists parallel freely, and the user can do the work more flexibly beyond a time-axis.

1. はじめに

設計の早い段階で様々な動作モデルを素早く試作するラピッドプロトタイピングは，大規模な製作を行う前段階の作業として有効である．プログラミングによりシステムの作成をする場合も，システムの設計を行った上で，作業者の意図した通りに一から組み上げていく必要があり，容易な作業ではない．しかしプロトタイピングを行うことで，設計や仕様の検証やシステムのイメージを具体化が行えるため，作業に着手しやすくなる．そのためラピッドプロトタイピングによって，敏速にプロトタイプを作成することは有意である．しかし，大規模なシステムの作成やプロトタイプの作成に関わらず，デバッグ作業はプログラミングにおいて必須作業であり，エラーの存在は，ソースコードをコンパイルし実行することで初めて発覚する．エラー箇所を特定するためには，デバッガを利用したり，Undo 機能を用い過去へ戻ったり，あるいはコメントアウトを行ったりしながら修正とコンパイル，実行を繰り返す作業を小刻みに行うのが一般的である．

複雑なシステムを開発する場合であれば，バグやエラーのないシステムを開発するために，デバッグ作業に時間を割く価値はある．しかし，プロトタイプシステムの作成は，あくまで，目的のシステムを開発するための前段階に行う行為であり，目的を達成するための事前準備でしかない．特にラピッドプロトタイピングにおいては，素早い開発を行うため，エラーの発見や修正などの試行錯誤に時間をかけるべきではなく，それに適した手法が必要であると考えた．そこで本研究では，ラピッドプロトタイピングによるシステムの作成に焦点をあて，時間軸の概念を取り入れたプログラミング支援を行う．過去のソースコードの状態と実行結果をパラレルに存在させ，視覚的に表示することでエラーやコードの入力ミスの発見を促進し，また，それらの任意の状態への自由な時間移動を可能とするシステムの試作を行った．

本稿は，以下のように構成される．次章で関連研究について述べ，3 章でデバッグ作業における問題点について，デバッグに用いられる機能の特徴と共に述べる．4 章でシステムの機能について述べる．5 章で時間とのインタラクションによるプログラミング手法について述べ，6 章でまとめを行う．

2. 関連研究

履歴情報を用いたコンテンツ制作システムとして，中小路らが提案している ART019 では，1 ストローク描画するごとに，その時間情報とサムネイルがリストに記録され，それを創作に活かすことが出来る[1]．太田らは，コンテンツを作る過程の操作などの創作時間そのものをリミックスすることによる新たなコンテンツ制作手法を提案した[2]．Chronicle では，過去の状態と使用したツールをサムネイルで表示し，作業工程を振り返りやすくするために作業時のパラメータをメタ情報として扱っている[3]．Autodesk Maya では，作業の履歴保存に加え，その時のパラメータを変更することで，作成している 3D モデルの対象部分を変更する機能がある．これらのように，時間情報を利用したコンテンツ制作システムは数多く存在する．他に，コン

† 明治大学理工学部情報科学科
Department of Computer Science, Meiji University
†† 独立行政法人科学技術振興機構, CREST
JST, CREST

ピュータ内での作業において時間情報を用いた研究として，川崎らの提案している Regional Undo では，スプレッドシートにおいて領域指定を行うことで部分的な Undo 機能を実現している[4]．Berlage が提案している Selective Undo では，履歴から操作を選んで Undo する手法が提案されている[5]．Prakash らはテキストエディタにおいて範囲指定を行い，その中で Undo を行うシステムを提案している[6]．暦本が提案する Time-Machine Computing ではコンピュータ環境の状態そのものを時間順に管理するという手法により，作業環境そのものを保存し，それらの時間移動を可能としている[7]．馬場らはビジュアルプログラミングシステムにおいて，木構造を用いた Undo 機能の提案をしている[8]．過去の状態を木構造で記録することで過去の状態全てへの時間移動が可能であり，俯瞰的に過去の状態を確認することができる．近藤らの Retrospector では，PC 内での殆ど全ての作業を記録し，10 秒おきにデスクトップ上の状態の画面イメージを作成し，作業者に過去の状態を振り返らせることで日常作業の支援を行なっている[9]．Lieberman は，ビジュアルプログラミング環境において，操作を時系列順に表示し，記録した内容をリアルタイムにフィードバックするための機能を持たせた[10]．

プログラミングに関連する研究として，本稿第二著者が提案している HMMMML では，プログラミングにおけるスペルミスや，セミコロン抜け等の単純なミスをコンパイラ側で超好意的に解釈するプログラミング環境を構築している[11]．本システムでは，コンピュータが自動的に誤りを発見しそれを好意的に解釈するものではなく，コンピュータ側が誤りのある箇所のみを提示し，作業者自身に誤りを気づかせることで，簡単なミスであれば頭を使わなくてもすぐに修正のできる環境の構築を行う．

3. デバッグ作業における問題点

ここでは，ラピッドプロトタイピングにおけるデバッグ作業における問題点と，Undo 機能，コメントアウト機能，ファイルのバックアップ，デバッガの特徴を述べる．一般的にデバッグ作業は，デバッガを使用したり，Undo 機能で過去の状態へ戻りながら，ソースコードの一部を書き換えたり，あるいはコメントアウトをしたりすることによってコンパイルと実行を繰り返し，エラー箇所を探す作業によって行われる．例えば，一度作成し，正しく実行できていたプログラムに新たな機能を付け加えた際に，正しく実行できなかった場合，Undo 機能により，正しく動作した時点の状態まで戻り，どのタイミングで動作しなくなったかを調べたり，新たに付け加えた部分のみをコメントアウトし，エラー箇所を探したりすることができる．

3.1 Undo 機能

一般的なエディタに備わっている Undo 機能は，一入力ごとに作業者の行った作業を一つ戻すことができる．しかし，修正箇所まで戻るためには，戻す必要のない部分まで消しながら戻る必要がある．更には，図 1 に示すように，一度入力修正を行うと，修正以前の内容は消えてしまい，Redo 機能を行なってもその部分へ戻ることはできないという問題点がある．消えてしまった部分は作業者自身が復元しなければならず，余計な手間がかかってしまう．また，戻るボタンもしくは Ctrl-Z キーを一入力ごとに一つずつしか戻すことはできないため作業者の意図した部分へ戻る場合，何度も Undo 機能を入力する必要がある．Ctrl-Z キーを押し続けることで連続的に戻ることはできるが，高速で Undo が実行されてしまう．以前動作した時点がどのタイミングで現れるかの判断ができないため，作業者の意図した位置で止めることが非常に困難である．

現状の Undo 機能には多くの問題点が存在する．しかし，その反面，過去の状態から，現在の状態までの行動であれば，全ての入力を記録してある．そのため，エラー箇所を発見するために，Undo を少しずつ行いながら，こまめにコンパイルを行うという，単純であるが手間のかかる作業を行うことで，以前コンパイルを行い正しく動作した部分まで，何も考えずとも必ず辿り着くことができる利点がある．そのため，必要のない箇所まで戻してしまう，一つずつしか戻れない，戻る位置の調節ができないといった問題を解決することができれば，プログラミングにおいて非常に有用な機能となりうる．

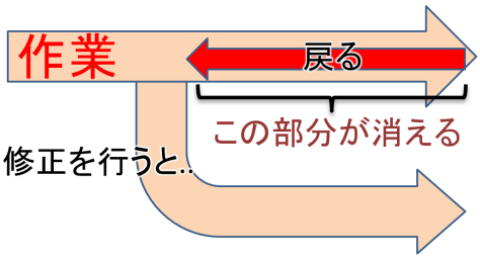

図 1　Undo 機能における問題点

3.2 コメントアウト

コメントアウトは，作業者が記述したコードを消してしまうことなく，無効化する手法である．そのため，Undo 機能のように必要以上にコードを消すことなく，部分的な修正を行うことが可能である．また，一度コードを作成したが，必要なくなってしまった部分をコメントアウトし，残しておくことで，後で使うことになったとしても，再度入力を行う手間を省くこともできる．コメントアウト機能は多くのプログラミング言語に存在し，その汎用性も非常に高い機能であるといえる．

デバッグ作業においてコメントアウト機能は，エラーの原因箇所を探索するために用いられる．記述したソースコード全てを，もしくはエラーの原因であると推測される場所のみを部分的にコメントアウトし，再度コンパイルをすることで行われる．しかし，コメントアウト作業は，作業者自身が任意の位置を指定して行うため，エラーがあると

思われる箇所について，あくまで推測でしかコメントアウトできない．そのため，コメントアウトを行った後に，正しく実行が行えるようになったとしても，そのコメントアウトを行った箇所すべての中から誤りのある箇所を見つける必要があるため，発見に時間を要してしまう．また，記述したコードが多いほど，コメントアウトをする箇所が増えてしまう．その場合，作業者はそのコメントアウトを行った箇所全てから，どこに何箇所あるか分からないエラーを探す必要があるので，ソースコードの作成時と同等，もしくはそれ以上に負荷の高い作業となってしまう．誤りの箇所を発見し修正を行ったとしても，（Undo 機能程ではないが）コメントアウトを外す復元作業の手間がかかってしまう．

3.3 ファイルのバックアップ

一度正しく動作した時のソースコードのバックアップをとっておき，それと比較を行うことは，デバッグ作業において非常に有効な手段であると考える．しかし，プログラミングにおいては，コードに変更を加えるたびにエラーが発生するか否かが変化してくる．正しく実行できたタイミングでバックアップをとることが一般的であるが，作業者自身は，プログラミング作業を一旦中断し，コンパイルを行わなければエラーがどのタイミングで発生したかを知ることはできない．そのため，バックアップを残しておいたデータは，エラーが発生する直前の最新のデータであるとは限らない．実行を行う直前に単純なミスをしてしまっただけで正しく動作しなかったとしても，最後にバックアップをとったものが，それよりも前の段階のものしかなかった場合は，比較対象としてあまり適切でないと考える．

3.4 デバッガの利用

デバッグを支援するツールとして，デバッガがある．デバッガにはソースコードの実行の流れを任意の位置で止めるブレークポイント機能や，止めた処理を 1 ステップずつソースコードの実行を行うステップアウト機能，ソースコード内で使用している変数の値を出力する変数確認機能などが存在する．デバッガシステムは，エラー箇所の発見や，バグの原因を探る際に有効であり，多くの開発環境に備わっている．本研究で利用している HSP[12]にもステップ機能や，変数の状態確認などが可能なデバッグウィンドウが備わっている．一方で，これらの機能を用いたデバッグ作業では，作成したコードや変数が正しいかを一行一行確認しながら進めることができるが，ラピッドプロトタイピングには向いていないと考える．

4. システム

4.1 システムの設計

前章のように，デバッグ作業時に行われる作業は複数あるが，それぞれ長所・短所がある．Undo 機能とコメントアウトを組み合わせて使用することも可能であるが，Undo 機能を使うことで，コメントアウトしていた部分も Undo されてしまうため，両方の機能を同時に使いながらデバッグ作業を行うことは困難であると考える．作業者自身が正しく入力をしているつもりでも，タイプミスをしてしまうなど，作業者の意図しないミスが発生してしまうことが多い．このような単純なミスがひとつあるだけでもプログラムは正しく動作せず，その都度上記のような作業を行いながらデバッグ作業を行う必要がある．このように，プログラミングにおけるデバッグ作業は，過去のどのタイミングでプログラムが正しく動作したかがはっきりせず，自分がどこまで戻る必要があるかを，Undo やコメントアウトをしつつコンパイルと実行をするという動作を繰り返しながら，手探りで探していくというような形で行われる．

デバッグ作業はどの手法を用いても，作業を一度中断し，エラーを探す作業を行う必要がある．それに加えて，Undo 機能のように一度行った行為を元に戻すような，時間の流れに逆らった作業を行うためにデバッグ作業は時間を要する作業となってしまう．

以上を踏まえ，ラピッドプロトタイピングのデバッグに必要なのは以下の項目である．

- 修正を行っても修正前の内容が失われない

ソースコードを過去の状態へ戻し，修正を行った場合でも，過去に戻す以前の内容全てを保持する．

- スムーズに時間移動の可能なユーザインタフェース

エラーが発生する前の段階で，作業者の移動したい位置を容易に判断でき，その状態を瞬時に復元できる UI.

- 実行を行わずとも結果を判断できる環境

作業者が意図したタイミングでコンパイルを行わずとも，コードの更新がある度に実行が行われ，その実行結果を確認可能な環境．

4.2 システムの実装

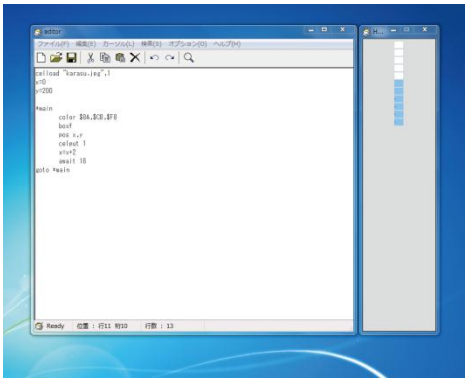

図 2　システム概要

本システムは，図 2 に示すように，実行結果とそのログをサムネイルとして表示する時間軸を超えて実行可能な「実行画面表示ウィンドウ」と，それに連動して時間移動可能な「ソースコードエディタウィンドウ」からなる．エディタウィンドウにコードが一行入力されたり，コードの

編集が行われる度に，その時点までのソースコード全体を記録し，同時にコンパイルを自動で行う．入力された時点でのソースコードが正しく実行可能である場合，実行結果をスクリーンショットで保存し，実行画面表示ウィンドウに実行結果のログとしてサムネイル表示する．サムネイルはコードの更新がある度に作成され，ログの一番下に追加されていく．また，ソースコードの記録を行うことにより，実行結果のサムネイルをクリックすることで，その実行が行われた時点でのソースコードの状態へ時間移動する．システムの実装は HSP で行った．

4.2.1 エディタウィンドウ

エディタウィンドウでは，入力が行われるたびにその時点でのコード全体を記録する．具体的な入力の判定は，任意の行に一文字以上の入力もしくは消去などの変更が行われた後に，その行からキャレットが他の行に移動したタイミングとする．コピーアンドペーストが行われた際は，ペーストが行われたタイミングで，カットアンドペーストが行われた際は，両方のタイミングでソースコードの記録とコンパイルを行う．また，コメントアウト機能によりコメント文が入力された場合や，何も入力されず，改行のみが行われた場合には実行結果に反映する変更は行われていないものとして，記録もコンパイルも行わない．

4.2.2 自動コンパイル機能

本システムでは，作業者自身が任意のタイミングでコンパイルを行わずとも，ソースコードが入力され，記録が行われると同時に自動的にコンパイルをする．図 3 に示すように，実行が正しく行われた場合は，その実行画面のスクリーンショットを，実行が正しく行われなかった場合は，その時点で入力された最新の行を赤く表示したエディタウィンドウのスクリーンショットをサムネイルとして，実行結果表示ウィンドウに作成された順に記録していく．また，実行できなかった場合のソースコード画像は実行できた場合のスクリーンショットよりも小さく表示する（図 3）．

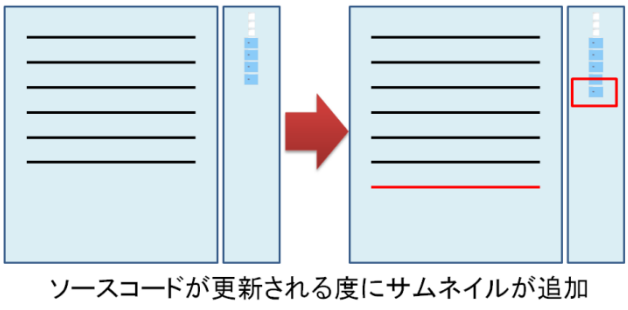

図 3　サムネイルの作成

4.2.3 実行画面表示ウィンドウ

図 4 に示すように，自動コンパイル機能で作成されたスクリーンショットは，実行画面表示ウィンドウに縦に並べられる．並べられたサムネイルは，マウスオーバーすることで拡大表示される．拡大表示されたサムネイルをクリックすることで，その時点でのコードの実行を行い，拡大された実行画面サムネイルの上に表示する．これにより作業者自身が実行を行わずとも，サムネイルに触れることで自分が意図した実行結果であるかをインタラクティブに確認することができる．

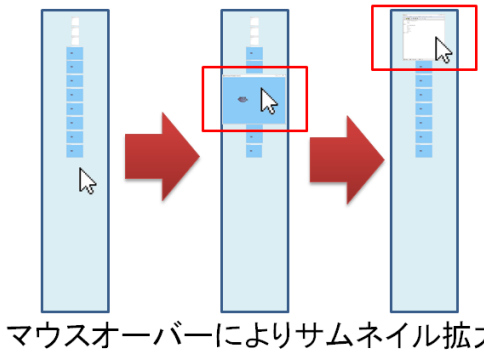

図 4　サムネイルの拡大

4.2.4 時間移動と分岐

実行画面表示ウィンドウにサムネイルとして記録された実行画面をクリックすると同時にソースコードエディタのソースコードを，その時点の状態へ戻す（図 5）．

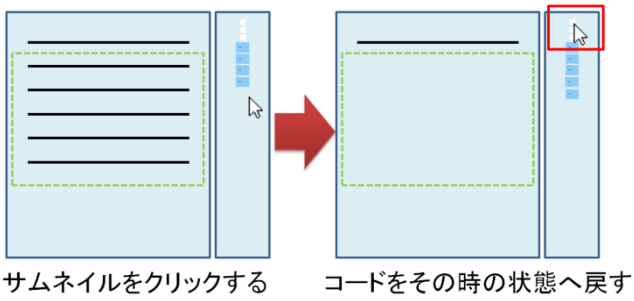

図 5　ソースコードの時間移動

ソースコードが過去の状態へ戻された状態でソースコードの修正が行われた場合，時間軸の分岐が発生する．過去の状態へ戻す前のコードと修正により作成されたコードは別の時間軸のものとして扱う．過去へ戻す以前の時間軸と区別するために，修正によって作成された実行結果のスクリーンショットは，変更前に最後に作成されたスクリーンショットの下に一定の間隔を空けて記録する．スクリーンショットは，常に最新の実行画面が一番下に追加されていく．また，派生元のサムネイルと新たに作成されたサムネイルを線で結ぶことで，修正されたコードがどのコードから派生したものかを視覚的に表現する（図 6）．

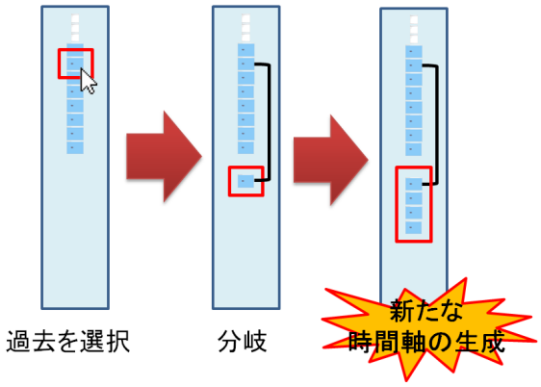

図 6　コードの分岐と派生元の表示

4.2.5 時間軸を超えたテキスト選択

図 7 のように任意のソースコードの範囲の選択をした状態で移動を行った際，移動した先にも同じコードが存在していた場合，範囲選択状態を保ったまま時間移動をする時間軸を超えたマーキング機能を持つ．過去のある時点でコードの範囲選択が行われた状態で未来へ移動した場合，同じコードが存在していたら，移動した先でも同じ部分に範囲選択が行われる．同じコードが存在しなかった場合は選択範囲の保持はされない．

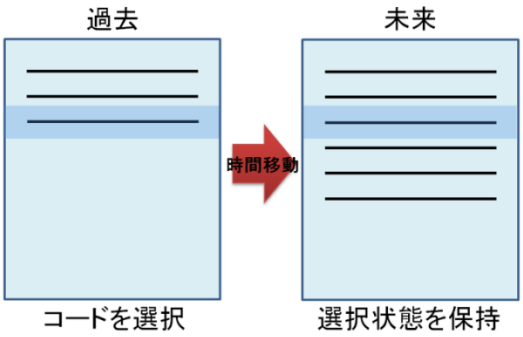

図 7　時間軸を超えた範囲選択

5. 時間とのインタラクション事例

本章では，本システムを利用した時間とのインタラクションによるプログラミング支援の事例について述べる．ソースコードの更新があるたびに実行が行われ，Interactive Ruby[13]や wonderfl[14]のように，実行結果を毎回確認しながら作業を進めることができ，コードを一行入力する度にその行にエラーが含まれているかどうかをリアルタイムに確認することができる．これにより，作業者はエラーが発生した直後に修正を行うことができ，エラー箇所を探索するという作業の削減に繋がる．今回利用した言語 HSP では一行入力だけで実行可能な命令が多く存在するため，特に有効であると考えられる．また，作業者自らが実行せずとも，自動的に実行され，実行画面のサムネイルが作成され，実行画面を比較することで自分が戻りたい過去の状態を判断することができる．エラーが発生せず，正しく実行されたとしても，サムネイルのクリックにより実行画面を開き，それを実際に触れることで，自分の意図した動作であるかどうかの確認をすることができる．

5.1 作業者の主観時間に沿った情報の記録

コードの作成や，修正を繰り返し，試行錯誤が行われると，作業が進むにつれて記録されたサムネイルが増え，実行画面群の表示が複雑になってくる．パラレルに存在するソースコードや実行結果を時間軸ごとに分け，木構造のように表示すると，横に広がり過ぎてしまうと考えられる．また，それらを自由に移動しながら作業を進めていくと，自分が行っている作業における主時間軸を見失ってしまう可能性がある．このような問題の解決法として，作業者の主観時間に沿って実行画面のサムネイルを追加していき，パラレルな世界を一軸上に表示するという方法をとった（図 8）．これにより，作業者は実行画面の違いと実時間のどちらをキーとしても，パラレルに存在する過去の状態の中から，任意の位置へ時間移動をすることが可能となる．

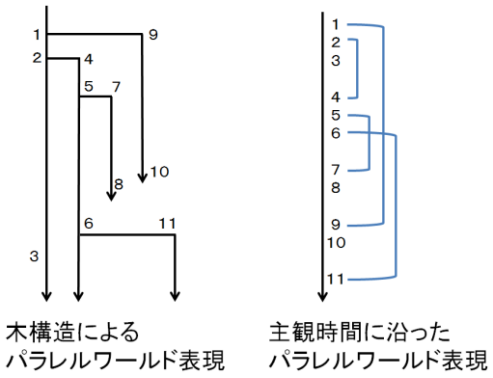

図 8　パラレルワールド表現の比較

5.2 分岐による試行錯誤

本システムでは，過去のソースコードの状態を全て保持しており，過去の任意の状態へ何度でも戻り，何度でも修正を行うことができる．そのため，デバッグを目的とした時間移動を行う他に，ある特定の状態から複数のプログラムを作成し，複数のパラレルなプログラムを見比べながら試行錯誤をしつつ作業を進めることも可能である（図 9）．通常の場合に，複数のプログラムを比較しようとすると，作成したプログラムを別ファイルとして用意し，それら一つ一つを順番にもしくは同時に実行する必要があった．いずれにしても，作業者自身が全てのファイルの実行を行う必要があった．このような手間も省くことで，よりシームレスな作業を行うことができる．

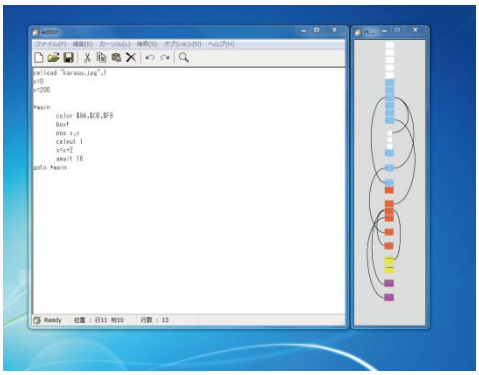

図 9　分岐による試行錯誤

5.3 時間軸を超えたテキスト選択の利用

サムネイルをクリックし，実行画面を表示することで，原因箇所を探すことができるが，その際，過去の特定の箇所以外を修正する必要がない場合も発生すると考えられる．例えば，エラーは発生していないが，実行結果が作業者の意図しない動作をした場合など，過去の特定の部分のみの修正を行いたい時などである．その場合，過去状態からの分岐を行う必要がないため，時間移動機能をエラー箇所の探索のみに使う．一度過去へ戻し原因箇所の特定を行ったら，また元の未来へ戻すという時間の往復をし原因箇所の修正を行うことで，分岐を起こさずに修正を行うことができる．しかし，過去へ戻すコードの量が数行に留まらず多数行あった場合，時間の往復をする際に過去へ戻った時に発見した箇所を見失ってしまう可能性がある．こういった問題に対して時間移動に依存しないテキスト選択によるマーキング機能が有効である．過去の状態へ戻り，エラー箇所を発見したら，その箇所の範囲選択することでマーキングを行う．その状態で未来の状態へ戻ることで，作業者は修正箇所を見失うことなく，エラー箇所の修正を行うことができる（図 10）．

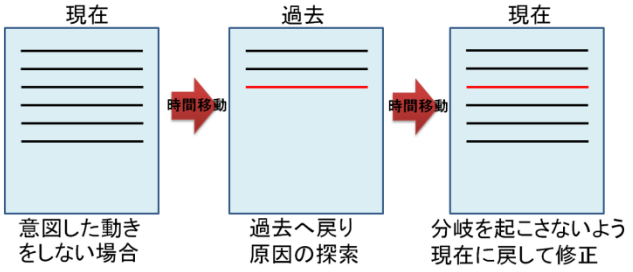

図 10　分岐を起こさない場合

6. まとめと展望

本稿では，過去のソースコードの状態への自由な時間移動と，その時点におけるソースコードの実行結果を表示することによるプログラミング支援システムを提案した．これらの機能を用いることで，作業者はエラー箇所の発見とその箇所の修正を容易に行うことが可能となる．現状ではソースコードが一行入力されるごとにコードの記録と実行を行なっているが，goto 命令の後にジャンプする先のラベルが必要であるように，複数の命令が同時に存在している時のみ正しく動作するものに対しての処理を行なっていないため，goto 命令が入力されてから，指定したジャンプ先のラベルが入力されるまでの入力が正しかったとしても，コードは全て実行不可能と判断されるという問題がある．これは，goto 命令とジャンプ先のラベルや関数などの一括りが入力された時点でコードの記録と実行を行うような処理を加えることで解決できると考えられる．

今後の課題として，今回は HSP のみに対応するシステムとして実装を行ったが，他言語にも対応したシステム実装も考えていきたい．また，本システムをより洗練させていくことで，ソースコードを記述してはデバッグを行うという小刻みにしか進むことのできない現在のプログラミング環境を改善させ，より良いプログラミング環境の構築が可能となるのではないかと考える．

参考文献

1) Y. Yamamoto, K. Nakakoji, Y. Niahinaka, M. Asada.ART019: A Time-Based Sketchbook Interface, Technical Report,KID Laboratory, RCAST, University of Tokyo (2006).
2) 太田佳敬，中橋雅弘，宮下芳明: 創作時間そのものを利用したリミックス，情報処理学会 ヒューマンコンピュータインタラクション HCI-148, No.2, pp.1-6 (2012).
3) Chronicle, http://www.autodeskresearch.com/publications/chronicle.
4) Y. Kawasaki, T. Igarashi: Regional Undo for Spreadsheets, ACM UIST Adjunct (2004).
5) Thomas Berlage: A selective undo mechanism for graphical user interfaces based on command objects.ACM Transactions on Computer-Human Interaction, 1(3), pp.269-294 (1994).
6) Atul Prakash and Michael J. Knister: A framework for undoing actions in collaborative systems. ACM Transactions on Computer-Human Interaction, 1(4), pp.295-330 (1994).
7) 暦本純一: Time-Machine Computing 時間指向ユーザインタフェースの提案, WISS1999 論文集, pp.55-64 (1999).
8) 馬場昭宏，田中二郎: ビジュアルプログラミングシステムのための UNDO 機能，日本情報処理開発協会 ビジュアルインタフェースの研究開発報告書, pp.176-187 (1994).
9) 近藤秀樹，三宅芳雄: 計算機上での活動履歴を利用する記憶の拡張システムの評価, WISS2005 (2005).
10) 宮下芳明，中橋雅弘: 学習者のモチベーション向上のための好意的解釈を行うフィジカルコンピューティング環境のデザイン，ヒューマンインターフェース学会論文誌, Vol.13, No.4, pp.303-313 (2011).
11) Henry Lieberman: Mondrian:A Teachable Graphical Editor, Watch What I Do-Programming by Demonstration, pp340-358 (1993)
12) HotSoupProcessor, http://hsp.tv/
13) InteractiveRuby, http://www.ruby-lang.org/ja/
14) wonderfl, http://wonderfl.net/