IPA

2012 年度 未踏 IT 人材発掘・育成事業　採択案件評価書

1. 担当PM

プロジェクトマネージャー：　原田　康徳　PM
（日本電信電話株式会社　NTT コミュニケーション科学基礎研究所　主任研究員）

2. 採択者氏名

チーフクリエータ：平藤　燎
（東京大学　教養学部理科二類）

3. 委託金支払額

1,792,000 円

4. テーマ名

CPU の理解を容易にするシステムと解説サイトの構築

5. 関連Webサイト

http://donut-lang.org/
http://ledyba.org/famicom/

6. テーマ概要

アンドリュー・タネンバウムによる著作「オペレーティングシステム－設計と理論および MINIX による実装」では、UNIX 風 OS である MINIX の設計と実装を通じてオペレーティングシステムやその周辺について実践的に学ぶ事ができる。「MINIX 本」との愛称で特に大学の教科書として広く世界で読まれ、リーナス・トーバルズが Linux を開発する動機にもなった。

本提案では、1980 年代に一世を風靡し誰もが知っている任天堂の家庭用ゲーム

機、ファミリーコンピュータ（以降、ファミコン）に着目し、それを再現するエミュレータを実際に開発しながら、コンピュータの動作原理や画像・音声処理を気楽に、しかも「わくわく」しながら学ぶことができる、「ファミコン版 MINIX 本」ともいうべきウェブサイトを作成する。さらに、非常にシンプルなビジュアル回路シミュレータを開発し、これを用いてエミュレーションで実装していた CPU を、実際の回路を読み解きながらシミュレーションで再実装する過程を体験できるようにし、学習者に対してソフトウェアとハードウェアの両方を垣根なく、ソフトウェア上のみでシームレスに理解を促すことを可能にさせる。

7. 採択理由

高レベルな API が充実してきたことで、一見すごいシステムを比較的簡単に作ることができるようになった。しかしそれがゆえに、コンピュータがどのように動いているのか、という地に足がついた基本的なところが遠のいてしまった。こういう提案が若い人から出てくるのは心強い。力のある人なので、独善的にならないように気をつけて、教育の専門家などの意見を聞きながら、世の中に本当に役立つようなものに仕上げていって欲しい。

8. 開発目標

2012 年のプロジェクトが開始した日本においては、Ruby on rails や JQuery を用いたウェブ開発や仮想機械上で動く Java を用いた Android 開発など、下層のソフトウェア・ハードウェアを抽象化したクロスプラットフォームな開発が活発である。インターネット上には様々な開発記事が充実し「学習の高速道路」とも呼ばれるほど簡単に学習していく事ができるようになった。初心者向けのプログラミング解説本などでも、JavaScript のようにブラウザで手軽に試せる環境を用いたものが多数出版され、以前に比べてプログラミングそのものの敷居は下がってきていると言える。

しかし、その一方で、その抽象レイヤの下に存在するバイナリやアセンブラ、バス、アドレス、入力、出力などのある意味では非常に基本的な部分に関する情報を見かけることは逆に少なくなってしまった。このままでは、抽象レイヤの上で動作するソフトウェアを開発するプログラマが生まれてきても、その下の抽象レイヤそのものを開発していくプログラマが生まれづらくなってしまうかもしれない。低レイヤを解説しようとする言説は現在でも少ないながらも存在はする。その対象としてよく選ばれるのが x86 のような現在圧倒的に普及したハードウェアを対象としたものと、CASL のような理想化された完全に架空のハードウェアである。前者は読者が毎日のように使っているハードウェアであり、題材とするには説得力があるものの、コンピュータの電源投入

から終了までのような根本的な所を余すこと無く解説するには複雑になりすぎており、あくまで教養程度の話題になってしまう。また、後者のような理想化されたハードウェアは理論などを論じるには便利であり、試験の題材とするには最適である。しかし、実際に動作するハードウェアは存在しないため、どうしても実感を持って学ぶことができない上、現実のハードウェア・アーキテクチャには存在するような、設計上の都合上生まれてくる一見奇妙な（だが必要であり妥当な）仕様などは当然存在しない。

そこで本プロジェクトでは、任天堂のファミリーコンピュータ、通称「ファミコン」を用いて教材を作ることとした。ファミコンは 1983 年 7 月 15 日に発売されたゲーム機で、CPU は Apple Ⅱ にも使われた 6502 を搭載し、比較的平易な命令セットながら、命令種類を減らすための幾つか奇妙な仕様が存在する。RAM も 2048 バイトと非常に小さい。そのような簡易なハードウェアでありながら、今でも根強いファンがたくさん存在し、「スーパーマリオブラザーズ」などの当時のゲームは今でも中高生にプレイされているなど、知名度は高い。また、関連特許もすでに切れており、法的な問題も存在しない。

本プロジェクトでは、コンピュータの動作中の動きや状態変化を簡単に可視化でき、動作に介入することも出来る「エミュレータ」と呼ばれる、コンピュータをコンピュータ上で再現するソフトウェアを用いて、バイナリ、アセンブラ、アドレス、バス、タイルグラフィックといった概念を理解しやすくするウェブサイトを構築する事を目標とした。

9. 進捗概要

当初、ウェブ上に JavaScriptを用いてファミコンを再現したエミュレータや可視化ウィジットなどを実装し、解説文を書きコンテンツを再生していく予定であった。 しかし、実際には2012年現在流行しているiOSやAndroidなどの「タブレット端末」において現実的な速度でJavaScript上のファミコンエミュレータを動かすことは不可能であったため、C++とOpenGLによるネイティブソフトウェアとした。その際、次のような技術的目標を設定した。

1.ファミコン実機と同じ60FPSで動作しつつ、可視化やエミュレータの動作への介入も行えること。

2.クロスプラットフォームであり、AndroidやiOSでも動くこと。

3.紙の本のように理解出来なかったところまでコンテンツを戻せること。

4.紙の本のようにいつでも「栞を挟める」こと。今回のシステムにおいては「セーブデータ」を作成して、それをロードすることですぐに状態を復帰できることを指す。

5.一人で膨大なコンテンツを作らなければならない事が予想されるため、コンテンツやウィジット作成の手間が少ないこと。

6.記事同士がWikipediaのようにキーワードや関連項目で繋がっており、自分の興味

や知識に合わせて自由に読み順を決められること。

今回の開発の結果、GUIツールキット「ちさ」とスクリプト言語「ど～なっつ」を開発し、その上に構築した教材を作ることができた。以下、開発結果の概要である。

1.PC上においては 60FPSを達成できたが、Androidタブレットnexus7においては GPUへの負荷が高く50FPS程度しか出ておらず、最適化が依然必要である。

2.OpenGL上でGUIツールキットを0から構築することで、AndroidとWindows/Linuxのクロスプラットフォームとすることが出来たが、iOSではコンパイラのC++の最新仕様”C++11”への対応が不十分であり、未だ動作できていない。

3.「ど～なっつ」の開発によって、分岐や任意の条件での繰り返しなどの複雑な状態遷移の必要なコンテンツのストーリーを、チューリング完全なプログラミング言語で自然に記述できるようになり、ユーザは理解できなかった所まで戻って読みなおす事もできるようになった。

4.「ど～なっつ」の言語処理系においては、セーブデータを書いてそれを読み戻す事ができるようになったが、周囲のGUIツールキットなどがまだ対応していない。

5.成果報告会用のスライドを既存のPowerPointと同程度の時間で今回のシステムで作成できた。コンテンツの作成しやすさの問題は解決していると考えられる。

6.記事同士のつながりを表すシステムは、まだそのための機構をある程度組み込んである程度で、実現できていない。さらに、今回作成したコンテンツエンジンを利用し、兵庫県立西宮香風高等学校で2013年1月18日に高校生を対象としたワークショップを行った。

開発内容

本プロジェクトはいくつかの試作の後にC++とOpenGLを用いたクロスプラットフォームネイティブアプリケーションとして開発された。その試作の過程も含めて記述する。

(1)JavaScriptファミコンエミュレータ

プロジェクト開始当初はタイトルの通りウェブサイトとするべく、JavaScrip上でファミコンエミュレータを開発した（図1）。

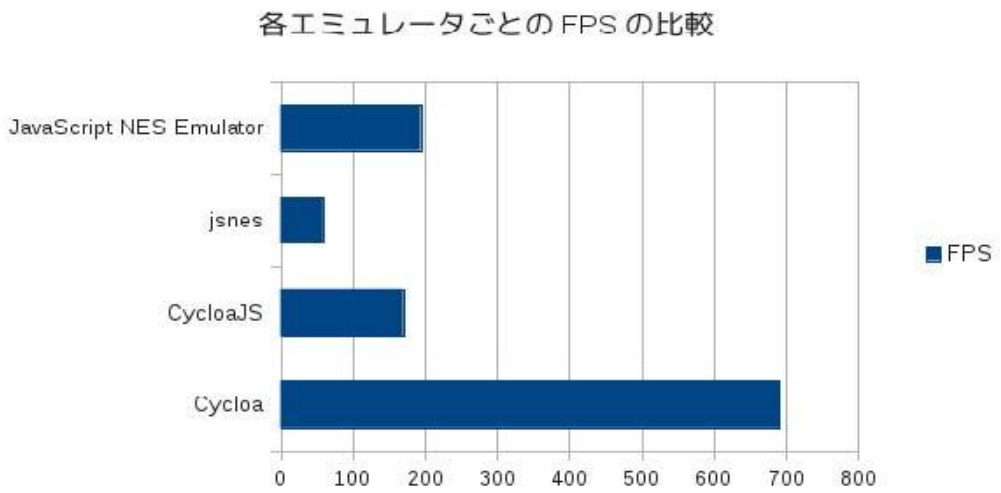

図1:ファミコンエミュレータ "CycloaJS"

既存の同様の試みとしてJavaScript FCエミュレータやjsnesなどが存在したが、前者は高速に動作するものの、グラフィックの処理において大きく省略している部分があり、解説したいと考えているプログラミングテクニック「スプライトダブラ」や「ラスタースクロール」などを再現しない。また、後者は再現度は高いものの、比較的最近のCPUにおいても60FPSが出ない程に低速である。今回はjsnes程度の再現度を保ちつつ、JavaScript FCエミュレータよりも徹底的に最適化を行うことで、タブレット端末においても十分な速度の出るエミュレータを開発することを目指した。

図2:エミュレータ同士のPC上でのFPSの比較（一番下のみ C++ネイティブ実装）

JavaScript FCエミュレータ含めた他の実装のソースコードや、ベンチマークを用いた実測などを重ねた結果、JavaScriptにおいて高いパフォーマンスを得るための手法は次のようなものが有効であると分かった。

·プロトタイプチェーンは使わない。具体的には _proto_プロパティを使わない。

·プロパティの検索を極力減らす。具体的にはobj.sthのようなネストした参照を何度も行わない。必要なら一度ローカル変数にコピーし、処理した後に書き戻す。

·スコープチェーンを浅くする。具体的にはクロージャ外への変数を参照しない。

·関数呼び出しは非常に重い。処理をインライン展開し、出来る限り一つの関数に処理を集約させる。

·多数の条件分岐があった場合、switch文とArrayに関数を入れてジャンプテーブルを作成する方法のどちらが速いかはブラウザに依存する。

得られた知見を手法を効率的に適用するため、コードジェネレーターを作成し、メモリの読み書き時のアドレスに応じた IOの振り分け（メモリマップドIO）をインライン展開するなど最適化を行った。結果、コンピュータ上においては JavaScript FCエミュレータほどの高速性を得ることは出来なかったが、 jsnesに対して3倍程度高速なエミュレータを作成することができた。

しかし、初代iPadにおいて同様のベンチマークを取ると、今回のエミュレータはファミコン実機での速度の約1/10程度(6FPS)でしか動作しなかった。

また、最適化を繰り返す過程で実機でデータバス・アドレスバスとエミュレータでの該当箇所の対応が曖昧になるなど、エミュレータを使って遊ぶのであればよいが、解説をするには向かない構造となってしまった。

(2)Google Native Client

速度の問題は明らかになったものの、依然としてWeb上での動作を目指していたため、Google Native Clientを検討した。これはJava AppletやAdobe Flashのようなプラグインを、 CやC++を用いて記述することが出来る実行環境である。調査として、C ++のエミュレータを移植（まだ未公開）した所、ほぼ通常のネイティブアプリと遜色のない性能が得られた。さらに、もう少し複雑なソフトウェアとして、Noiz2saというBSDライセンスの弾幕2Dシューティングゲームも移植して公開した（図3）。

図3:Noiz2sa for Chrome

移植による調査の結果、通常のファイルシステムではなくスレッドをまたいだコールバックを多用した独特な方法を使用する必要があり、他環境との両立したプログラムの作成が難しいことが判明した。

Native Client上のアプリの使い勝手はFlashやJavaAppletよりも高いとは言えず、Web、しかもPC上のChromeでしか動かない現状では、他プラットフォームを諦めるデメリットはあまりにも大きい。NativeClientは断念した。

(3)Android向け Javaアプリ／iP hone向け Objective-Cアプリ

Webを諦め、Android/iOS向けのアプリとして開発することを目指した。

以前 Androidで開発をした経験から、Android上のJavaで実装されたエミュレータをファミコン実機と同じ速度で動かすことは恐らく可能であるが、それには相当の最適化が必要だと予想された。C/C++を使うこともできるが、Javaと連携させる場合、JNI(Java Native Interface)を用いる必要があり、言語を跨ぐためデバッグも困難になる。

iOS向けの開発では、非常に個人的な事情であるが、MacOSXを使って開発していると、マウス腱鞘炎になり開発が続けられなくなる事が判明していた。

二つのターゲット向けに開発することのコストも問題となる。Android用Javaアプリや iOS向けのObjective-Cアプリを開発することは現実的選択とは言えなかった。

(4)C++と OpenGLによるネイティブアプリケーション

最終的にはC++とOpenGLを用いたネイティブアプリケーションとした。この組み合わせは AndroidとiOSはもちろんのこと、PC上でも動作する。PC上であれば充実した開発ツールを用いることができ、開発効率が高い。ひとまずPC上で動作するものを作成した上で、AndroidやiOSなど向けに後で移植する戦略を取った。

(5)技術目標

ほぼゼロからの新規開発となったため、いくつか技術的目標を設定した。

(a)リアルタイムでファミコンエミュレータが動作すること

ファミコンの動作を伝える周辺のウィジットなどもすべて同じ速度でリアルタイムに動作することを絶対の条件とした。

さらに本システムはある種の「電子書籍」のようなものを目指していたため、紙の本で可能であるにも関わらず既存のシステムでは出来なかったことを実現し、さらに紙の本では不便であった所も改善することを目標とした。

(b)読んだところ以前の任意の所まで戻って読み返せること

例えば数学の本であれば、論理の展開が理解できなければ、一旦戻って理解できているところから再度、読みなおす事ができる。小説ならば、人物関係がわからなくなったらその人物が初登場するところを読み返せばよい。非常に基本的な本の「機能」である。

しかしソフトウェア上ではそのような事が必ずしも出来るとは限らない。ソフトウェアがそのための何らかの仕掛けをもつ必要がある。

(c)状態をセーブし、そのデータを用いて復帰できること

紙の本ならば「栞」を挟むことで、どこを読んでいてもそこで中断し、その後にまた再開することができる。しかし、ソフトウェアの動作を途中で中断させ、その後全く同じ状態から再開することは一般に容易でない。OSごとハイバネートさせるような方法もあるが、時間が掛かり、制限も大きい。

(d)レイアウトを無視して矢印や線を引けること

印刷された本の上に、矢印や線を引き、自分なりに注釈をつけることで、理解の助けとする人は珍しくない。また、 P DFに対してそのような注釈を行えるソフトなども存在する。

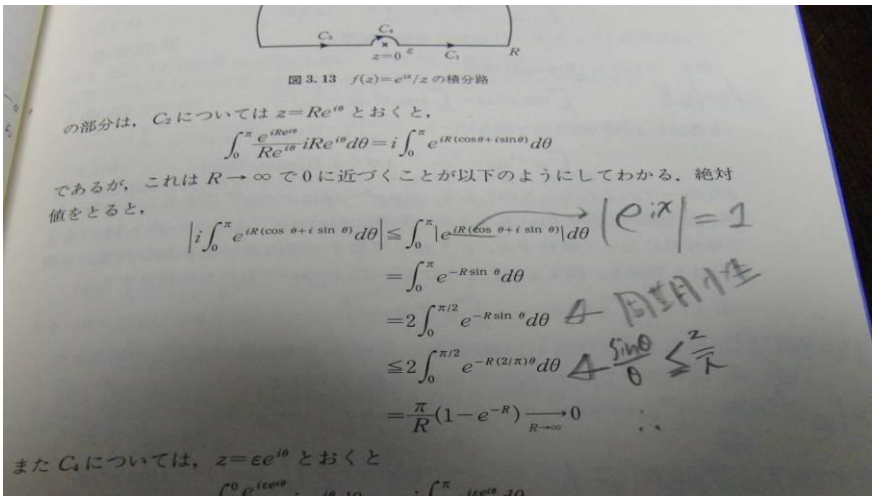

図4:数学の式に自分で注釈をつけた例

今回のシステムでは、そういった線や矢印をあえて筆者側から引いて「今説明しているのはどの事なのか」をわかりやすく説明するようなシステムが出来ないかと考えた。

伝統的な GUIツールキットでは各ウインドウやウィジットが画面上の一部分を専有しているという構造で、その上にさらに描画を行うことは不可能ではないが、パフォーマンスの問題や入力イベントのハンドリングに気を配らなければならないなど、問題がある。

(e)必要な画像や脚注を、すぐに示してくれること

紙の本において、本文で示している画像や表を何度も説明に使用するために、違うページを何度も参照しながら読み進めなければならないケースがある。 同様の例として、脚注が全て後ろに纏まってしまっているため、何度も本文を中断して後ろの方のページを開かなければならないこともある。これらは紙の本の物理的限界であるが、コンピュータにおいては平易に乗り越えられる問題であると思われた。

(f)適当にページを開いて興味の赴くままに読み始められ、そこからさらに広げていけること

暇つぶしとして本を読む際に、本のページを適当に開いて関心のある内容であればそれを読む、という方法がある。小説のように長いコンテクストがある場合は不可能であるが、短い記事の集合となっているような本であればそのような読み方が可能である。今回のコンテンツも短い記事の集合となるのではないかと予想されたため、そのような読み方をこのシステムでもサポートすることを目指した。

さらにWikipediaのように緩やかに独立した記事同士がネットワークを作り、本文中の

リンクや関連項目を辿って読み進められるようにすることで、適当に読み始めた地点から自分の興味や知識に合わせて読み進められるシステムを構築することも目標とした。

(6)開発環境

以上の目標を決定した所で、開発環境の構築を行った。今回は Windows 7、Fedora 17、さらに VPS上の Cent OS 6の3つの環境を同時に使用した。

午前中は Windows上で、午後は Fedora上で開発し、さらにソースコード共有サイト「github」上のリポジトリにソースコードがpushされると、Cent OS上でビルドと単体テストが自動で行われるようにセットアップした。このように同じパソコン向けプラットフォームとはいえ3つの環境向けに開発とチェックを行い続けることで、後々にAndroidや iOSへ移植する際の問題点を軽減する事を狙った。

実例として、C++11の標準ライブラリstd::atomicをFedoraで使用した翌日にWindows上ではまだ動かないと判明し、標準ライブラリとほぼ同じインタフェースを提供する boostの実装と切り替えられるようにしたケースなどがある。こういった措置はWindows上だけでなく、後に Androidに移植する際にも効果的に働いた。

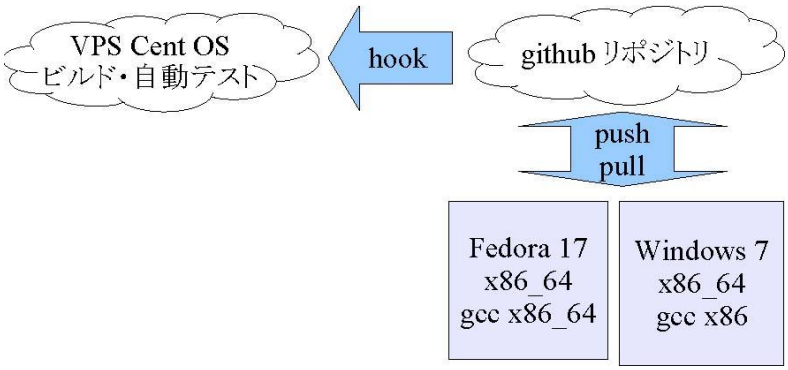

図5:今回の開発環境

(7)GUIツールキット「ちさ」の開発

まずはベースとなるGUIツールキット「ちさ」を設計・開発した。先述までのようにパフォーマンスを稼ぐために、OpenGLネイティブのGUIツールキットとした。

最終的ターゲットはタブレット端末であったため、AndroidやiOSのツールキットを手本として設計・実装を行った。

XMLでレイアウトを記述する近年では一般的な設計となっているが、今回のプロジェクトのために幾つか特殊な機構も実装した。

(a)複数の Worldを持ち、いつでも切り替えできる

Worldとは、一つの画面構成を表す、Androidでいうところの Activityのような存在である。ただしActivityと違い、スタックで管理されておらず、タブのように瞬時に切り替える事を前提として設計されている。

今回のシステムにおいては、一つの記事を一つのWorldに割り当て、複数のWorldを立ち上げる事で複数の記事を同時に読めるようにした。Wikipediaの関連項目を複数タブで開いて気の向くままに読むような動作をこのシステムでも実現できる。しかし、本プロジェクト期間内ではまだ実際には使用できていない。

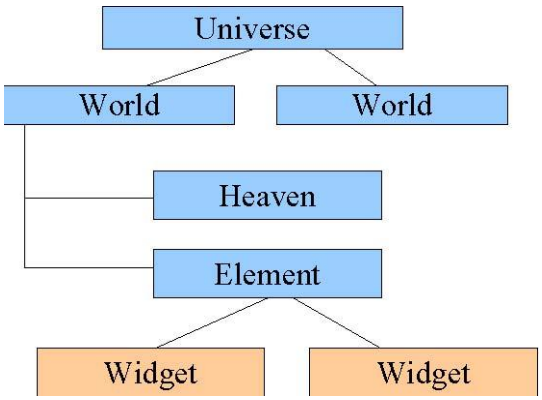

図6:「ちさ」におけるGUIの構造

(b)ウィジットとレイアウトの一部分離

通常、 GUIツールキットでは全ての画面構成要素を一つのクラスのサブクラスとすることが多い。AndroidであればView、SwingであればComponentである。しかし、今回のGUIツールキットでは、通常のボタンやタブ、それらを纏める通常の機能を担う部品を「Element」と呼称し、画像やエミュレータの画面、メモリビューワ、説明文などのコンテンツを伝える役割を担う部品である「Widget」と分離した。Widgetは専用の「 WidgetElement」の下に配置され、少し特別扱いを受ける。

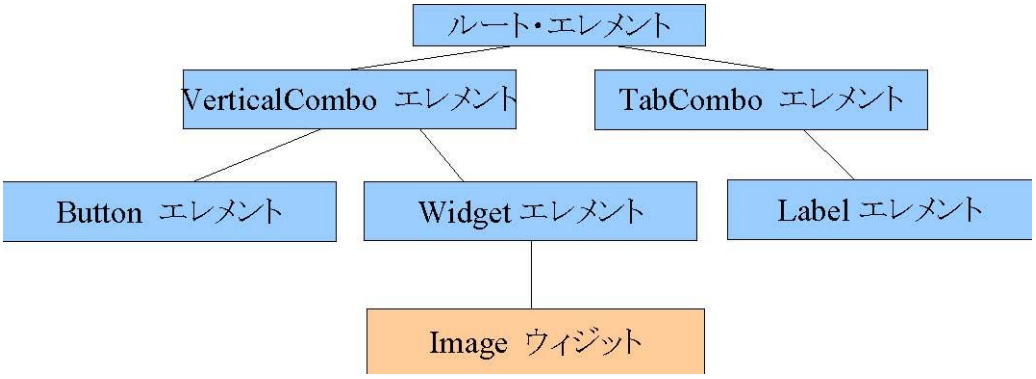

図7:Elementを用いたレイアウトの例

ElementはAndroidのレイアウトのように一旦ロードしたらそのまま固定されたりせず、別のElementの定義をXMLから読み込む事で、動的にレイアウトを変化させる事ができる。

この時に、すでにある古いElementの下にあるWidgetを新しいElementへ選択的に引き継げる仕組みを作成した。これも記事を書く際にわかりやすくするために使えるのではないかと考えているが、本プロジェクト期間内ではまだ使用できていない。

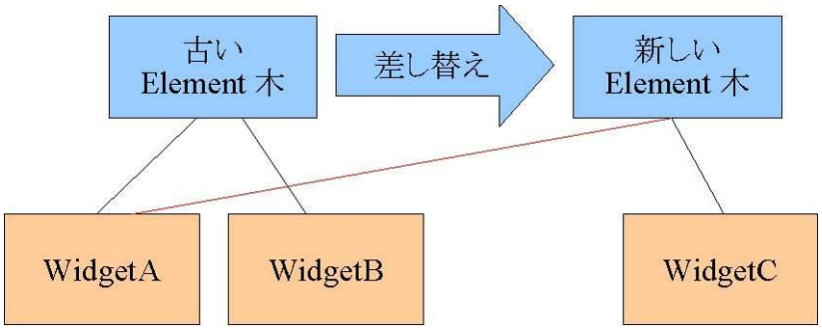

図8:WidgetAを新しいレイアウトへ引き継いだ例

(c)オーバーレイ表示のための専用領域

技術目標としていた「レイアウトを無視した矢印や線を用いた注釈」をつけるためのデータ空間として、「Heaven」を用意した。HeavenはWorldごとに存在する空間で、Worldを切り替えればHeavenも同時に切り替えられる。Elementでは通常のGUIツールキットのように各Element要素やその下のWidgetが矩形をテリトリーとして占拠する構造になっているが、Heavenでは画面上の任意の位置に部品を配置することができる。

その際に描画位置を指定するための機構として、「Angel」を作成した。Angelを使うと、ElementやWidgetの画面上での位置を追跡し、その位置に対して線を引いたり矩形を表示したり下線を引いたり、それに紐付けて新たなElementを配置することができる。

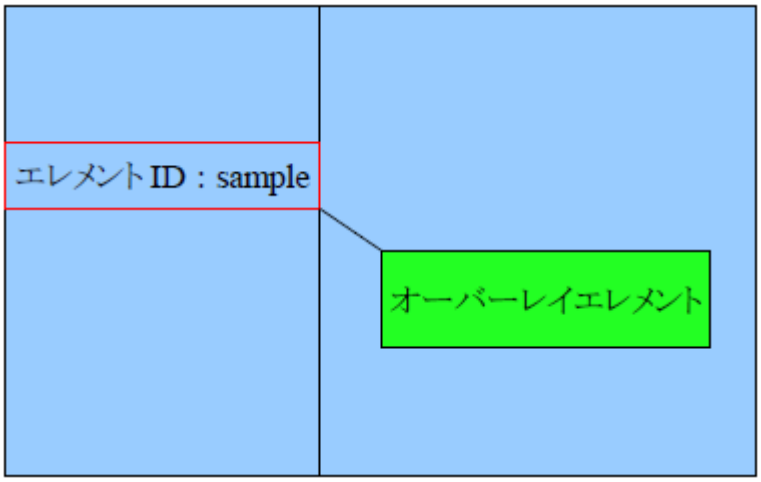

図9:エレメントID：sampleに対して矩形とオーバーレイエレメントを表示した例

(d)大量の文字をレンダリングするための専用ライブラリ

今回のコンテンツでは、メモリの中身や機械語の実行状態などを表示するために文字を大量に、しかも頻繁に内容を変えてレンダリングする必要がある。

そのための専用ライブラリがあり、OpenGLテクスチャ切り替えの最小化や頂点転送の効率化などを自動で行う。このライブラリによって、レンダリングの性能が数倍以上改善された。

(8)シナリオ制御の設計

このGUIツールキット「ちさ」上で「電子書籍」を作るためのシナリオ制御機構について、いくつか検討を行った。

(a)XMLやJSONなど

GUIツールキットでも用いているXMLなどを用いて、今回のコンテンツの文章や画像の表示順序を管理する方法を検討した。既存のライブラリなどを用いる事で容易に読み込み処理などを実装できる。再生時のコンテキストが少ないため、技術目標とした、任意の時点におけるセーブや、コンテンツを元に戻すことなども容易に実装できると思われた。

しかし、ユーザのアクション（例えば、コンテンツ内で与えられた課題をクリアした、あるいはできなかった、あるいは課題を飛ばして次に進む事を選択した）に応じて説明を分岐したり、スキップしたりといった複雑なシナリオの制御のためにはアドホックな機能をいくつも実装しなければ対応できず、コンテンツ作成の手間が恐らく非常に大きいであろうことが予想された。エミュレータのような複雑な構造を制御する機構を、XMLのような単純なデータだけで記述できるとも考えにくい。

(b)Lua/Squirrel/JavaScript/Python

ユーザのアクションに対し、エミュレータと連携してシナリオを進めていくには、チューリング完全のコンピュータ言語を用いるのが適当であると思われた。

アプリケーションに組み込むための軽量なスクリプト言語として、LuaやSquirrel、JavaScript、組み込むにはやや規模が大きいがPython、などが存在する。どれも非常に強力な言語で、特に前者の2つはゲームなどのリアルタイムが要求される分野でも活用されている。これらを用いたコルーチンとしてシナリオの制御を組むことが出来れば、 (a)で検討したデータで書く際の問題を解決できると考えた。

プログラミング言語は実行途中の状態からそれ以前の時点の状態にまで戻るような操作（以下これを時間操作と呼称する）はできないため、目標としていた任意の時点までコンテンツの状態を戻す操作は直接的には実現できないが、実行状態をセーブデータとして読み書きする事ができれば、それを用いて間接的に時間操作を行うことが可能である。 しかし、調査した所、これらの言語の実行状態をセーブデータとしてファイルに読み書きする方法は存在しなかった。

(c)Scheme/Haskell

SchemeのcallccやHaskellのContinuation Monadを用いると、処理系の任意の時点での実行状態と、その先の計算を取得して操作できる。これを継続」という。この「継続」を用いてWebアプリケーションの状態遷移をコルーチンで書くことができるフレームワークが、「Kahua」などいくつか存在する。これらを手本として SchemeやHaskellなどを用いてシナリオ制御を実装することを検討した。

「継続」を用いれば、処理系の指定した時点まで処理を巻き戻し、そこから処理を再開することが可能である。ただし、処理系の外の環境までは巻き戻す事が出来ない。このためか、継続をWebではなく処理系外への副作用の多いGUIに応用した例を発見することが出来なかった。

また、SchemeやHaskellなどにおいても処理系のセーブデータを読み書きする方法はやはり存在しないようだった。

(d)ニワン語

ニワン語はニコニコ動画内で使うことができる特殊なスクリプト言語である。この言語のソースは、ニコニコ動画に投稿された動画のコメントとして表現され、再生が投稿された動画内時間に差し掛かった瞬間にその処理が発動する。このスクリプト言語を使ったゲームなども作成されている。

動画は任意の時刻へシークする事ができるが、それに応じてニワン語の処理状態

もシークされ（たように少なくとも見え）る。このスクリプト言語は今回のシステムで必要とされている要件のうち「時間操作」の要件を（おそらく）満たしているが、処理系は完全にクローズドであり、実際にはどうなっているのか知る術がない。以前、ニワン語の挙動から互換処理系「ねこまた」を開発した経験があり、様々なソースを用いて挙動を検証したが、たまに過去の状態と未来の状態が混ざる（シークに失敗する）事があり、ますますよくわからない。このため、「ねこまた」では順方向の処理しかサポートしていない。 言語処理系外の環境、例えば投稿されたコメントの表示や動画などに関しては、ニコニコ動画のプレイヤに「ニワン語」が実装される以前からシークすることができたため、少なくとも言語処理系外の環境に関してはそちらを用いて戻していると思われる。

(e)既存の処理系に「時間操作」と「セーブの読み書き」機構を実装する
これまでの検討において明らかになった既存の言語における問題点は、
·「時間操作」ができないか、出来ても操作できるのは言語処理系の中だけである。
·処理系のセーブデータを読み書きする機構がない。
の2つである。
　これらの2つを解決するため、当初SquirrelやLuaなどの比較的小規模な言語処理系を改変して解決する事を検討した。
　しかし、調査の結果、どちらも比較的小規模とはいえ複雑な構造を持っていた。掛かる工数の見積りは難しく、途中で致命的な技術的問題に直面する可能性もあった。
　そこで、今回はスクリプト言語をゼロから設計することとした。手間は掛かるが工数は予測でき、プロジェクト期間内で完成させることができると思われた。

(9)スクリプト言語「ど～なっつ」
　以上の検討を受けて、新たなスクリプト言語「ど～なっつ」を次の技術目標を達成できるように設計・実装した。
· 言語処理系の任意の時点に処理系の状態を巻き戻し、その後さらに現在の状態にもう一度戻す、「時間操作」が行える。
· 言語処理系内の環境だけでなく、「反副作用」によって言語処理系外の環境も時間操作の対象とする
· 言語処理系の任意の時点での処理状態を記録してセーブデータを書くことができる。
以上のための実装について記述する。

(a)言語処理系内での「時間操作」
　言語処理系の実行の状態を特徴づけるのは、少なくともど～なっつにおいては以下の 3つである。

・オブジェクト内の変数の状態

・コールスタック・計算スタック

・プログラムカウンタ

ど～なっつ内での時間を管理するために、各ど～なっつインスタンスは 1つだけ擬似時計を持っている。この時計は新たなソースを読み込んで実行するか、ソフトウェア割り込みから復帰するか、言語内から「ほむら」と呼ばれる特殊オブジェクトの特定のメソッドを呼ぶ事でカウントアップされる。ど～なっつの処理系は、この擬似時計の各時刻での状態を基準として時間操作を行う。プログラムのソースコードの 1行単位で巻き戻したりは出来ないが、そこまで粒度の高い時間操作は必要にならないと考えられる。

ど～なっつにおける「オブジェクト」はJavaScriptのそれを模しており、各オブジェクトは名前に紐ついた「スロット」を保持している。さらに、各スロットは擬似時計での時刻と値のペアのリストである。値にはどのような種類のオブジェクトも入れることができる。

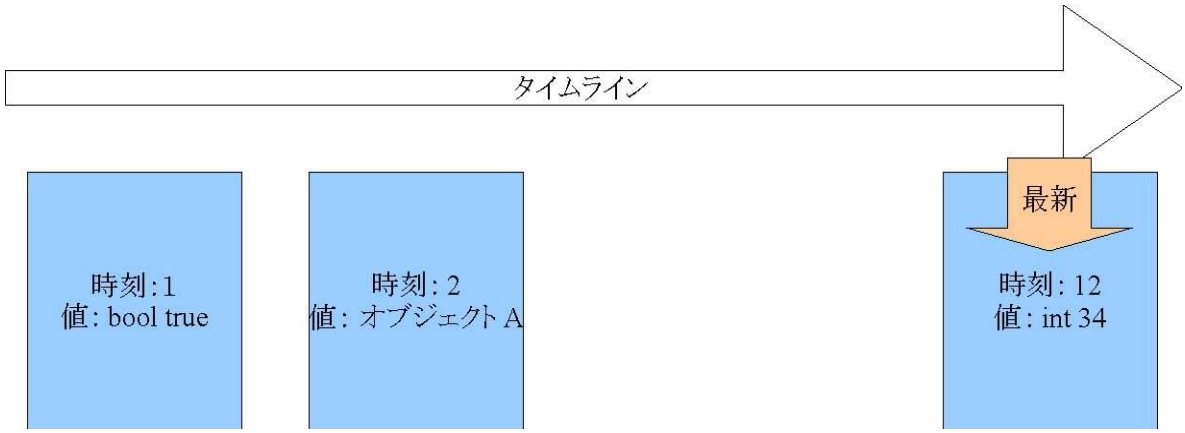

図10:とあるオブジェクトのスロット一覧

各オブジェクトのスロットへ値が代入されるとき、まず現在の時刻を問い合わせる。その時刻とスロットの持つ最新の値の時刻を見比べ、同じであればその最新と値とすり替え、そうでなければ新しい時刻と値のペアを作り、リストの末尾に追加する。たとえば図10であれば、現在の時刻が 12であれば新しい値(例えばオブジェクト B)で古い値である「 in t:34」を上書きし、そうでなく例えば現在の時刻が 13であれば、新しく「時刻 13->オブジェクトB」というペアをリストの最後に付け加える。

時間操作で過去に戻った場合、その時刻以降の値を全て捨て去る。ただし実際に捨て去るのは戻った後に何かしらのソースを実行した時なので、実行していない場合はそのまま現在の時刻などの他の時刻に再度時間操作を行い続ける事ができる。

ど～なっつに於いては、ローカル変数やグローバル変数などのスコープ・システム

も全てオブジェクト内のスロットとして実現されため、オブジェクトの状態を操作することでローカル変数などもすべて時間操作することができる。

(b)「反副作用」による言語処理系外の時間操作

言語処理系がその外界に影響を与えるのは、言語処理系が C++などの外側の言語で記述されたネイティブ関数を実行し、「副作用」を発生させる事による。

計算機科学において「副作用」とは、コンピュータの状態を何かしら変化させるものを全てを抽象的にさすが、その中身をもっと具体的に表現すれば「ウインドウを開く」「画像ウィジットの中身を差し替える」などの行為である。抽象的に「コンピュータの状態を全て完全に元に戻す」ことは一般的に難しいが、これらの具体的な行為として捉えれば「ウインドウを閉じる」「画像ウィジットの中身を差し替え以前の物に戻す」などすることで元に戻すことが出来る。

今回「ど～なっつ」において、そのような具体的な「副作用」を戻すための操作を「反副作用」と呼称し、時間操作が発生したときにその反副作用を適用することで、処理系の外の環境も元に戻す事ができる。

副作用を発生させるメソッドの戻り値は複数の値のタプルであり、そのうちの一つが反副作用である。つまり、反副作用の生成とその正当性の証明を行う 責任はプログラマに任される。

各オブジェクトは、処理系外への副作用に対応した反副作用を、時間の矢に沿って、オブジェクト内のスロットと同様に時間と反副作用のペアのリストとして保持している。

時間を巻き戻す際には反副作用を適用して同時に「反・反副作用」を生成し、以前の反副作用と置き換える。反・反副作用は概ね元の関数呼び出しと同じ副作用を表す。この反・反副作用もプログラマの責任で生成するので、元と同じ副作用でなくとも構わないが、元と違うものにする意味を現在まだ見いだせていない。

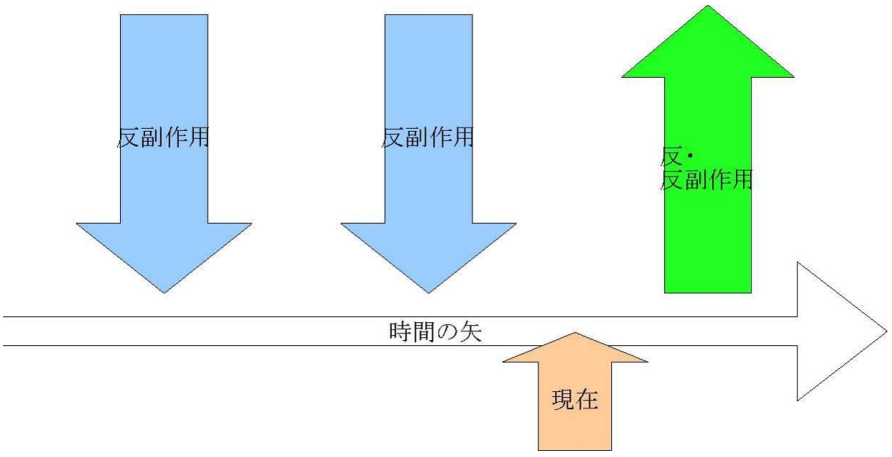

図11:とあるオブジェクトの反副作用と反・反副作用の状態

この図11のような状態でプログラムを実行した場合、スロットの時と同様、現在時刻以降の反・反副作用を全て捨て去るが、この状態でプログラムを実行しないまま現在に戻ってくる場合は、反・反副作用を適用して得られた反・反・反副作用（≒反副作用）で反・反副作用を再度、差し替える。

後々別の使い方が発見された時のために、毎回反・反副作用や反・反・反副作用を生成しているが、そうでなくても構わないかもしれない。

なお、副作用によっては反副作用を生成できない事もある。例えばネットワーク経由でメールを送信するなど、ネットワークが関わってくると途端に反副作用を生成することは難しくなる。そういった場合は反副作用を生成しなければ、現在時刻以前の反副作用の内容やスロットの内容を全て捨て去り、それ以前の過去に戻れないようになっている。

反副作用を用いずに時間を巻き戻す際によく使われるイディオムとして、「Mementoパターン」がある。このパターンでは、巻き戻したいオブジェクトの全状態を保持しておき、巻き戻す際にその保存した全状態の情報を用いて戻す。このMementoパターンのような方法にくらべ、反副作用のアプローチでは副作用で変化した部分だけを記録しておけばよいため、メモリ効率がよく、実装に掛かる工数も少なくなる。ただし、今回のプロジェクトではセーブデータの保存も行うために、全状態の保存も行えるようにはなっている。

(c)時間をスクリプト言語内から操作することが出来る

スクリプト言語内にある特殊オブジェクト「Homura」とそのメソッドを通じて、上記の時間操作をスクリプト言語内から発生させることも可能である。簡単な例としては過去に戻ると処理系の状態が全て戻る事を利用し、ループを作成することが 出来る。

実例として図12のようなソースを挙げる。実行すると、図13のような結果が得られる。

```
1. //普通の変数は、時間操作の影響を受けますが…
2. tabeta=0;
3.
4. //Homuraは時間操作前の事を覚えていて、時間操作の影響を受けません。
5. Homura.counter=0;
6.
7. //このあとの命令を使って、この時点まで時を戻せるようになります。
8. save_time=Homura.tick();
9.
10. if(Homura.counter < 10){
11.   // ドーナッツを食べましょう。
12.   // この足し算の結果は、時間操作で戻ってしまいます。
13.   tabeta++;
14.
15.   // ドーナッツを食べた回数を表示しましょう
16.   System.println(tabeta, "番目のドーナッツを食べた！");
17.
18.   // Homuraに入った値は時間操作の影響を受けないので、
19.   // 時間操作を行なってもこの足し算の結果は戻りません。
20.   Homura.counter+=1;
21.
22.   // 上で記録した時間まで、スクリプトエンジン全体を戻します。
23.   Homura.seek(save_time);
24. } else {
25.   //ど～なっつでは、else節を省略することはできません。
26. };
```

図12:サンプルソース

```
1. % donut time_op.donut
2.   1番目のドーナッツを食べた！
3.   1番目のドーナッツを食べた！
4.   1番目のドーナッツを食べた！
5.   1番目のドーナッツを食べた！
6.   1番目のドーナッツを食べた！
7.   1番目のドーナッツを食べた！
8.   1番目のドーナッツを食べた！
9.   1番目のドーナッツを食べた！
10.  1番目のドーナッツを食べた！
11.  1番目のドーナッツを食べた！
```

図13:実行結果

(d)コールスタック・計算スタックの時間操作

コールスタックや計算スタック、プログラムカウンタなどは全時刻に対して保持しており、この情報を用いて時間操作を行なっている。

(e)言語処理系のセーブデータの読み書き

言語処理系のセーブデータも、ど～なっつの動作状態を特徴づける 3つ、オブジェクト、各種スタック、プログラムカウンタのそれぞれのセーブデータから構成されている。

このうちプログラムカウンタのセーブを作成するために、プログラムの実行形態として仮想機械を選択した。実行形態として一番実装工数が掛からないのは生成した抽象構文木をそのままトラバースして実行する方式であるが、抽象構文木のトラバースの途中の状態でのセーブデータを作成することは難しい。構文木のトラバースで実行処理を実装する場合、そのために C++の再帰関数を使うが、多段に再帰した C++の実行スタックの状態を保存することは不可能である。

一方、仮想機械を用いれば、プログラムカウンタの数値をセーブし、ロード時も数値をそのまま読み出せばそのまま仮想機械の状態を復元し、実行を再開することができる。

このために命令種類数が 20種類程度の簡易な仮想機械命令セットを設計し、コンパイラとインタプリタ方式の仮想機械を実装した。仮想マシンのアーキテクチャには大きく分けてレジスタ型とスタック型の二種類が存在するが、実行コンテキストが小さくなり実装も簡単なスタック型のアーキテクチャとした。

(10)バイナリから目覚めるぼくらのファミリーコンピュータ！

「ちさ」と「ど～なっつ」をベースとして実際にファミコンを題材とした教材を作成した。今回は後に述べるワークショップの教材としての機能のみを実装している。

図14:ワークショップで用いた教材

左上にはエミュレータのゲーム画面があり、その下には実行するプログラムを選択するためのボタンが配置されている。Lesson1とLesson2は今回のプロジェクトで作成したゲームであり、それ以外は他者の作成したゲームで、それぞれ許可を得たかGPLやMITライセンスの自由なゲームであるため、使用上法的な問題ない。

さらにその下にあるのは、ファミコンのCPUである 6502の命令実行の様子である。クリックすることで各命令を書き換えることも可能である。もちろん、適切に書き換えなければプログラムがクラッシュしてしまうが、その際は右下の「戻る」ボタンを押せば「ど～なっつ」の時間操作によって書き換える直前の状態に巻き戻す事ができる。ステップ実行も可能であり、この時にど～なっつの時間操作を使うと、ステップ実行をステップごとに戻す事が可能である。

図15:命令書きかえとメモリ書き換え

上段の右側はメモリ空間である。6502の持つ16ビット、64KBのメモリ空間（$0000から$FFFF）を全て表示している。クリックすることで、命令と同様にメモリの内容を書き換えることができる。

上段の真ん中と下段は「メモリ検索」機能のための領域である。ゲーム内の残機やキャラクタの座標などの数値を、64KBのメモリ空間のうちの RAM部分である先頭2048バイトの内容から検索することができる。一番下の「候補をリセット」を押すことでその候補をリセットし、2048個の候補からスタートする。左側の「値を検索」を使うと、入力した値に対して、「同じ」「違う」などそれぞれの条件で候補の中からその条件に合うものを絞り込む事ができる。この絞り込みは再度リセットされるまで続くので、もう一度値を検索すれば、絞り込んだ中からさらに絞り込む事が可能である。

左側は「記録メモ」による比較である。残機やスコアなどの具体的な数値は先述の機能で探せるが、座標などの具体的な数値はゲーム画面だけからは分からない。「記録メモを取る」で一旦候補の値を全て記録し、その記録メモでの値と比較して「変わった」「変わってない」などの条件にあうアドレスを絞り込むことで、間接的に探すことができる。

例えば座標であれば、一旦立ち止まって「記録メモ」を取り、そこから適当に歩いて「変わった」ボタンを押せば、そこで絞り込まれる候補の中に座標が入っているだろう。こちらの比較も、値を検索する際と同じで、候補をリセットするまで絞込みが続く。何度か繰り返すことで、現実的な候補数にまで絞り込む事ができる。

(11)兵庫県立西宮香風高等学校でのワークショップ

原田PMがワークショップについて Facebookで告知を行ったところ、兵庫県立西宮香風高等学校の松本吉生先生が興味を持ってくださり、2013年1月18日に同校でワークショップを行った。

高校生の男性生徒の方が6名、松本吉生先生と井下貴朝先生に参加して頂いた。生徒の方は一人を除きプログラミングなどを行った事のある方は居なかった。

まずは実際にファミコン実機を目の前で分解し、 CPUや RAMのチップと、今回のシステム上の画面の対応を示し、その中身が表示されていることを伝えた。

図16:ワークショップの様子

(a)Lesson 1

最初に、 Lesson 1として簡単なプログラムを実行してもらった。これは非常に単純なプログラムで、メモリ$0000と$0010を足して$0020に代入しつづけるだけの単純なプログラムである。メモリの書き換え機能を使って$0000と$0010の値を変更してもらい、その値が確かに$0020に代入されることを確かめてもらい、確かにコンピュータが計算機である所を体感してもらった。

さらに、足し算命令である ADCをSBCに、キャリアフラグをクリアする CLCをセットするSECに変えることで、引き算を行うように変更することが出来ることも体験してもらった。ただし、6502は命令数を削減するために純粋な足し算・引き算命令はなく、キャリアフラグの影響のある ADC/SBC命令しか無かったため、分かりづらかったようである。

図17:Lesson 1

```
$80b8 ea        NOP
$80b9 ea        NOP
$80ba ea        NOP
$80bb 18        CLC
$80bc a5 00     LDA[$00](=01)
$80be 65 10     ADC[$10](=02)
$80c0 85 20     STA[$20](=03)
$80c2 4c bb 80  JMP$80bb(=18)
$80c5 00        BRK
$80c6 00        BRK
$80c7 00        BRK
$80c8 00        BRK
$80c9 00        BRK
$80ca 00        BRK
$80cb 00        BRK
```

図18:Lesson 1の本体

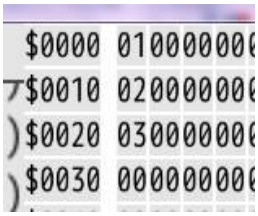

図19:足し算を行った結果

(b)Lesson 2

単純な足し算からレベルを上げ、もう少し複雑なプログラムである Lesson 2へと続けた。Lesson 2は非常に単純なドット絵お絵かきプログラムである。カーソルを移動しAボタンやBボタンを押すことで、画面上にドットを出現させ、ごくごく簡単なお絵かきを行うことができる。

図20:Lesson 2

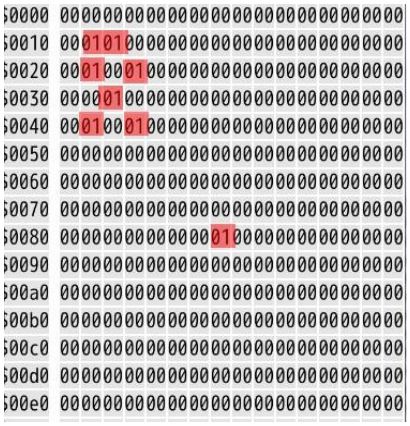

図21:メモリ空間上での値とドット絵が対応している

このドット絵は、$0000から$00ffまでの16x16バイトに書き込まれた内容が画面用RAM(CPUとは別のアドレス空間にある)に展開されて表示される仕組みになっている。
まずはその事を伝えずにドット絵を適当に書いてもらい、その後右の画面に注目してもらって、その事になんとなく気づいてもらった。

メモリ上の値を任意に変更すれば、その事がすぐに画面上に反映され、ドットが白くなったり黒くなったりする。その事を通じ、画面上に表示されるキャラクタも数字によって全て管理されている、ということを体験してもらった。

さらに、このプログラムでは、ドットを描画する位置をカーソルで指定する。
そのカーソルの位置も、メモリ上の値として管理されているはずだと伝え、どこにあるか探してもらった。ここでメモリ検索機能を説明しようと考えていたのだが、このプログラムはカーソルの座標以外殆どレジスタのみを使って書かれており、正解のアドレスも$100と$101であったため、目視ですぐに分かってしまった。もっと不慣れな人が多く、出来る限り簡単にして「メモリを検索する」という行為そのものを伝えやすくしようと考えていたのだが、その必要は無かったようだった。

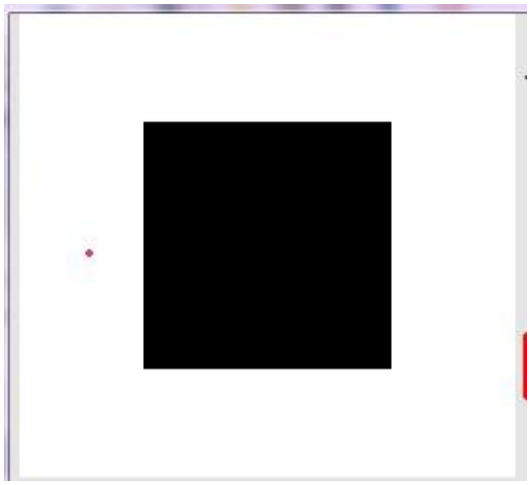

図22:座標アドレスを直接書き換え、画面外にカーソルを飛び出させた

そのカーソルの値を書き換えることで、通常黒い四角形の中から出ることができないカーソルをその外へ飛ばす事ができ、そうすると今度は矢印キーを押しても戻ってくる事ができなくなる(”バグる”)事を体験してもらった。

(c)実際のゲームで自由に書き換え

単純なプログラムでのレッスンはここまでとし、その後は他に搭載している実際のゲームを用いて自由に書き換えてもらった。

図23:ゲームによってはおかしな値になると警告することも

さきほどのようにキャラクタの座標を書き換えてキャラクタを壁にめり込ませたり、残機を増やしたりといった私が直接的に伝えたことを行った方も居れば、そこからさらに前オブジェクトとの全接触判定をすり抜けさせたり(どうやったのか聞いたところ、分からないとのこと)、通常は表示されないはずの警告メッセージを表示させたりなど、私の伝えた事以上の様々なゲーム上での操作を行なう生徒が現れた。

図 24:キャラクタが柱の中に

今回、スクリプト言語を実装して「元に戻す」操作ができるが、ワークショップという形態ではあまり活用されなかった。最初からやり直せば大体事足りたようであった。

(d)ワークショップの反応

ワークショップを受けてもらった生徒からの感想では、「理論的には分かっていても実際には試した事がない内容だったので勉強になった」など好意的な反応を得た。先生側からも、Lesson 1/2のような簡単なゲームを自作すれば他の先生も活用できるのではないかといった助言を得ることができた。

(12)Androidへの移植

今回の教材を最終的なターゲットとしていたAndroidタブレット「nexus 7」へ移植した。

図25:nexus 7で動作している様子

移植にはAndroid NDKと呼ばれる、Androidアプリケーション上でC/C++コードを実行できる仕組みを用いている。AndroidではOpenGLではなく、組み込み環境向けのOpenGLESを用いる必要があり、OpenGLと比べて関数の名前が一部異なっている等の差異があるため、その吸収を行う必要があった。

まだ 50FPS程度ででしか動作していないが、CPUの負荷は十分低いため、GPUへの負荷が大きすぎるのだろうと推測される。まだ最適化が必要である。

(13)iOSへの移植

現在まだiOSへの移植を行えていない。i OSでの開発においてはclangというコンパ

イラで開発を行う必要があるが、このclangがまだC++の最新仕様C++11に完璧には対応しておらず、今回のソースコードをコンパイルすることが出来なかった。しかしclangの開発は活発であり、時間が解決する問題であると考えている。

10. プロジェクト評価

コンピュータのハードウェアを直接触れるかのようなシステムを提供して、そこからコンピュータについて学んでもらう、というアイデアはやはり本物であった。PM との議論で、様々な課題を出したが、それらに対して、素直に受け入れるか、反論するかを、きちっとプログラムを書いて確かめてから判断していた点に非常に好感が持てる。その積み重ねで、非常に安心できるものができた。彼の開発能力は極めて高い。

与えた大きなハードルとしては、実際に生徒に使ってもらって、そのフィードバックを得るということであった。これによって、自分だけしか使えないようなシステムではなく、きちんと、変な操作をしても落ちないような、安定したシステムを作らなければならない。この目標は早目に設定し、それに向かって、いくつかの条件をクリアしながら実験の準備などを行った。この時期までに、これができていたら、その出来具合で判断して、こういった実験に協力していただけそうな学校を探す、というやりかたである。無事に授業も成功し、担当してくださった先生や実際に受けた生徒にも好評であった。いろいろな要望も頂くことができた。

最終的に出来上がったものは、それらの課題をクリアしただけで、すでに十分なのにもかかわらず、簡単に時間軸を巻き戻すことができるプログラミング言語というものまで開発してしまったのは、度肝を抜かれた。このプログラミング言語がどれくらいのインパクトがあるのかは、正直よく分からない。これが開発のメインテーマであったら、たぶん採択はしていなかっただろう。

ではあるが、これまでの平藤君のこだわりと、完成度の高いものを作る能力からすると、数年後に化けるような、すごいアイデアなのかもしれない。

11. 今後の課題

本人はプログラミング言語の面白さの方に興味が移りつつあるが、その前に、ここだけはしっかりとやってもらいたい部分としては。このツールは先生方が自由に教材を作ることができて、それによって、コンピュータの様々なことをわかりやすく学ぶことができるものになっている。マニュアルや例題、教材を交換するプラットフォームやコミュニティなどは、ある程度までは平藤君自身がやっていく必要がある。ここまでやると一つの区切りをつけることができる。ぜひ、それを実行して、コンピュータ教育の重要な教材の一つにまで成長させて欲しい。