アプリで未来をつくるスマートフォン・ワールド

Interface増刊

Smartphone World

Volume. 4

http://smart.cqpub.co.jp/

twitter account / @sw_cq

1冊まるごとAndroid!

ラクしてゲームを作る!

Android & iPhone 開発テクニック

Corona SDK×Luaで物理演算/enchant.jsで3Dゲーム
PhoneGapでADKプラグイン/Unityを使ってみる
プロが使う開発環境でゲームを作るには

第2特集

Android×マイコン・ボード スーパー実践レシピ

AndroidとArduinoで楽器アプリ
温度/湿度/照度メータ
音声認識で動かすクラウド扇風機
ラジコン戦車をIOIOボードで動かす
RT-ADK miniでロボット雲台を作る
NDKでGPSを使ったARアプリ

CQ出版社

ゲーム開発の歴史と最近の傾向

ゲームと開発とスマートフォン

田中 宏幸

スマートフォンで人気のアプリと言えば，ゲームです．ここでは，そのゲームの進化の歴史をエンジニアの視点から振り返るとともに，最近のゲーム開発事情を紹介します．

写真1　ファミリーコンピュータ

皆さんは，近頃のスマートフォンは家庭用ゲーム機のスペックと似ていることをご存じでしょうか？

●開発の進化

私が始めてプログラムしたゲーム機はファミリーコンピュータ（以下ファミコン）（写真1）でした．ファミコンの開発言語は6502アセンブラというシンプルなアセンブリ言語です．当時私はBasicとC言語しか知らず，これがプログラム言語！？と，途方に暮れた記憶があります．ただし，パソコンとは違い，ファミコンには絵を簡単に画面に出せる「スプライト」という機能があり，やはりゲーム機はすごいと関心したものです．

その後，ハードウェア本体が進歩する（表1）につれて，開発環境も変わってきました．PlayStation（写真2）の開発言語ではCがメインになり，PlayStation2（写真3）の開発時にはC++がメインになりました．C++はオブジェクト

表1　ゲーム機のスペックの移り変わり

年	機種名	CPU	CPU動作周波数※1	メインメモリ
1983	ファミリーコンピュータ	2A03	1.79MHz	2KB
1988	メガドライブ	68000＋Z80A	7.67MHz/3.58MHz	64KB
1990	スーパーファミコン	5A22（65C816互換）	最大3.58MHz※2	128KB
1994	PlayStation	R3000A	33.8688MHz	2MB
1996	Nintendo64	R4300カスタム	93.75MHz	4.5MB
1998	Dreamcast	SH-4	200MHz	16MB
2000	PlayStation2	R5900拡張	294.912MHz	32MB
2001	GameCube	PowerPC Gekko	485MHz	40MB
2002	Xbox	Mobile Celeron(Pentium IIIベース)	773MHz	64MB（VRAM共有）
2005	Xbox360	PowerPC カスタム（3コア）	3.2GHz	512MB（VRAM共有）
2006	PlayStation3	Cell（PowerPCベース）	3.2GHz	256MB
2006	Wii	PowerPC G3 750CLベース	非公開	88MB

※1　この数値の大小が直接ハード性能を表すものではないが参考までに掲載する
※2　動作周波数を1.79MHz/2.68MHz/3.58MHzの3段階で切り替え可能

GameCube

写真2 PlayStation

写真3 PlayStation 2

指向言語のためソースの再利用性が上がり，また多人数での開発も可能となりました．

開発環境もどんどん改善されてきました．例えばPlayStation3やPSVita，Xbox360はVisual Studio2010で開発できるようになり，またハードウェアのメーカーが様々なパフォーマンス測定ツールを提供してくれています．Xbox 360では，DirectX SDKでおなじみのPIXが提供されており，頂点数やポリゴン数，フレームバッファ，シェーダーの負荷などが一目でわかります．もちろんPlayStation3やPSVitaでも同じようなツールが提供されています．

開発環境が豪華になるにつれてクオリティも上昇していきます．例えば，昔はプログラマが一人でゲームの開発を行うのが主流でした．今では各分野の要求事項が高くなり，描画処理や物理演算，AI，モーション，ネットワークなどプログラマの役割が専門化しています．

また，昔はいわゆる「勝ちハード」「負けハード」というものが明確でしたが，今はそういった境界がありません．例えば**Xbox360**（**写真4**）は海外で売れており，**PlayStation3**（**写真5**）は国内で売れているという状況のため，複数のハードでゲームを出すというマルチプラットホーム開発が主流となってきました．

●ゲームエンジンとスマートフォン

このような状況から，今のゲーム開発は「エンジン」ベースで行われるのが主流です．このエンジンは社内で制作されているエンジン（内製エンジン）と社外で制作されていて購入するエンジン（外製エンジン）の大きく2種類に分かれます．

有名な内製エンジンだと，カプコンの「MT Framework」やスクウェアエニックスの「Cristal Tools」，バンダイナムコゲームズの「NUライブラリ」などがあります．

また，外製（サードパーティ製）エンジンはEpic Gamesの「Unreal Engine」やCrytekの「CryEngine」，シリコンスタジオの「OROCHI（**図1**）」もここに分類されます．

日本での基本的な制作方法は，各分野の専門化たちがエンジン部分を制

写真4 Xbox360

写真5 PlayStation 3

図1 OROCHI（http://www.siliconstudio.co.jp/middleware/orochi/）

写真6 iPhone4S

写真7 F-07D

写真8 EeePad TF201

作し，ゲームの遊びの部分や，特殊な表現部分はタイトルごとに作成するというスタイルです．エンジン部分はある程度共有化できるし，タイトルごとにエンジンを書き換えるといった柔軟な開発も可能です．エンジンのクオリティがタイトルのクオリティに直結します．

逆に海外では，一流のエンジンを購入し，そのエンジンの枠の中でゲームを構築します．柔軟性は失われますが，「現実をシミュレートする」ようなタイトルとの相性は良く，エンジンの開発リスクを負わずに済むというメリットもあります．

このまま国内のゲームメーカーは内製エンジン，海外は外製エンジンで開発するのが主流となると思いきや，海外から黒船がやってきました．**スマートフォン**です．

米Canalysが発表した資料によると，昨年中の全世界のスマートフォン出荷台数は4億8770万台で成長率は68%とのことです．デスクトップパソコン，ノートパソコン，ネットブック，タブレットの合計である4億1460万台よりも上回っています．

また，国内の2011年度のスマートフォン出荷台数は2330万台と言われており，これは売上が好調なニンテンドー3DSの500万台を遥かに上回っています．そして，スマートフォンの中で一番多いジャンルは「ゲーム」です．

この状況をゲーム業界各社が見逃すはずはありません．こぞって参入を試みましたが，ここに大きな壁がありました．それは，開発言語とハードウェアの多様性です．

●スマートフォンの壁

ゲーム業界の使用言語は長らくC++のみでした．しかし**iPhone**(**写真6**)はObjective-Cで，**Android**(**写真7**，**写真8**)に至っては主にJavaです(NDKでCも使える)．もちろん開発環境も「Xcode」や「Eclipse」と，今までのゲー

図2 Unity3D(http://unity3d.com/unity/)

図3 Unreal3d(http://www.unrealengine.com/)

写真9 PSVita

写真10 iPad

表2 PSVita, iPhone, iPad, TF201の仕様比較

	PSVita	iPhone4S	iPad2	TF201（参考）
CPU	Cortex-A9（4コア）	Cortex-A9（2コア）	Cortex-A9（2コア）	Cortex-A9（4コア）
GPU	SGX543MP4+（4コア）	SGX543MP2（2コア）	SGX543MP2（2コア）	CPUに内蔵（12コア）
メインメモリ	512MB	512MB	512MB	1GB
VRAM	128MB	メインメモリと共有	メインメモリと共有	メインメモリと共有
画面解像度	960×544	960×640	1024×768	1280×800
タッチパネル	前面＋背面	前面	前面	前面
ジャイロセンサ	○	○	○	○
加速度センサ	○	○	○	○
カメラ	前面＋背面	前面＋背面	前面＋背面	前面＋背面

ム開発の手法とは大きく異なります．

また，iPhoneはハードの種類が少ないですが，Androidはさまざまなハードウェアや OSバージョンがあります．Androidは解像度だけでも320×240から1920×1080まであり，アスペクト比すらバラバラです．

このような状況では，新規にエンジンを制作するにも，内製エンジンを移植するにも期間やコストが非常に掛かります．ゲーム業界が得意とする重厚なタイトルがなかなか制作できず，かといって軽いタイトルの開発では競合タイトルとの差別化も難しい状況でした．そのため，既存のタイトルの移植が主流になっていました．

しかし，ここに来てまた大きく流れが変わります．スマートフォンを得意とする外製エンジンの登場です．

まず筆頭はUnity（図2）です．その機能と低価格さから瞬く間にゲーム業界に広がりました．今ではゲーム業界で圧倒的なシェアを誇ります．また，Unrealエンジン（図3）もiPhoneに対応し，同エンジンを使用したゲーム「Infinity Blade」は2300万ドル（約18億円）の売り上げとなりました．これ以外にもALCHEMYやngCoreなど，さまざまなエンジンが揃いつつあります．

●PSVitaとスマートフォン

また，PSVitaとのマルチプラットホーム開発という話も出てきています．実はPSVita（写真9）はiPhone4SやiPad2（写真10）と，とても近いスペックです（図4）．

CPUやGPUは同一型番であり，違いはコア数が4か2かの差程度です（表1）．画面解像度もほぼ同じで，双方ともタッチパッドとジャイロセンサ，加速度センサを搭載しています．Objective-CでもC++のコードを書けるので，ある程度ソースコードの流用も可能です．

もし今後UnityがPSVitaに対応すれば，スマートフォンとコンシューマのマルチプラットホーム開発はさらに加速するでしょう．

スマートフォンが現れた当初は様々な問題のためゲーム業界の強みを生かしたタイトルがなかなか出せない状況でした．しかし，優れた外製エンジンが多数登場した事により，今後は培ったノウハウを生かしたタイトルが多数出るのではないかと期待しています．

田中 宏幸
Hiroyuki Tanaka
株式会社イリンクス代表取締役社長兼プログラマー．ゲームを作った事のあるハードは，ファミコン，PC-9801，Windows，GAMEBOY ADVANCE，PS，PS2，PS3，Xbox360，Wii，フィーチャーフォン，iPhone．現在はPS Vitaと格闘中．

プロユースのゲームエンジンALCHEMY実践例

「BAROQUE」iPhone移植レポート

ここでは，本誌p.82～に解説しているツール「ALCHEMY」について，その実例を紹介する．PlayStation2用ゲーム「BAROQUE」をiOSへ移植した例である．

株式会社スティング

世の中には各分野ごとにさまざまなミドルウェアがあるが，ALCHEMYはゲームのために作られたミドルウェアである．

このようなミドルウェアの存在意義は，いかに同じリソースをたくさんのプラットホームに対応させることができるかということであるが，このALCHEMYは対応プラットホームが多い．最近のものだとWindows，PS3，Xbox360，Wii，そして今流行のiPhone，iPadやAndroidにも対応している．改めてカタログを見てみるとLinuxという表記まである．一瞬ゲームの需要があるのか？とも思ったが，組み込みLinuxで動いているものは意外と多いので，そのうち使うことになるかもしれない．

そして，ゲーム機用のミドルウェアがiPhoneやAndroidに対応したというのは重要であり，いままではゲーム機で動いていたものをそれらに移植するためのハードルがぐんと下がったことになる．事実，iPhoneのノウハウがほぼない状態だったにもかかわらず，ALCHEMYが対応したのでPS2のゲームを移植しようか，ということになった．以下にそのとき感じたことを簡単に書いていこうと思う．

今回対象になったゲーム「BAROQUE」は既にWiiに移植されていた（こちらもALCHEMYが対応したので移植されたもの）こともあったので，ある程度ソースの整理もついており，移植作業自体はおおむね順調に進んだ．そして，作業開始から程なくして画面が表示されるようになった．それも初めてのプラットホームでありがちな適当なテスト用のオブジェクトなどではなく，実際にPS2で動いていたゲームの画面だ．

あまりにも簡単にiPhoneの画面に実物が表示されてしまったので多少拍子抜けしてしまったくらいだ

この"画面が早い段階で表示された"

というのがiPhoneへの移植では大変有効であった．

というのも，iPhoneにはコントローラが存在しない．入力は画面のタッチで行わなければならない．そこで画面に仮想コントローラを表示すると，今度はその部分が指で隠れてしまうという問題が発生した．画面の下には色々な情報が表示されていることが多いので，それをどこに表示すればよいのか考えなければならなかった．このあたりは実際に触りつつ適切な位置へ調整をしなければいけない部分なのだが，実画面が早い段階で表示されていたことで，色々と試すことができたのだ．

そして，タッチパネル周りはプログラマが新しくコードを書き起こしたのだが，そのための時間を確保することもできた．

今回，PS2版の開発はALCHEMYのバージョンが4.0だったが，iPhone対応版では7.0になっていたので，かなりやっかいな問題が起きるのではないかと懸念はあった．しかし開発元(シリコンスタジオ)のサポートによって，特に大事にもならず作業が進んでいった．

実際，ミドルウェアを使っていると，それ自体の細かい使い方が分からなかったり，足りない機能があったりすることがある．その場合，開発元へ質問なり要望なりを出すのだが，海外製だとやり取りが英語になり，なかなか意思の疎通が難しい．そこへゲーム関連の専門用語が混じってくると，なおさらである．その点，ALCHEMYは日本語で質問や要望が出せるので安心である．

そうしたほかのものへの労力がかからない分，パフォーマンスの向上に集中することができ，トラブルらしいトラブルにも見舞われず，無事App-Storeの審査も通過することができた(現在は公開を休止している)．

結果，プログラマは最初の研究などの工程を飛ばしても十分iPhoneのノウハウをためることができた．デザイナやサウンドの人員もほぼさくことなく，iPhone特有の問題のみに注力することができるという，ほぼ理想的なミドルウェアの運用ができた

ほかに今回の移植作業では出番は無かったが，主要なモデリングツールとのデータのやり取りが容易なのをはじめ，各種データ形式が利用可能であるのは便利だ．パフォーマンス測定ツールや最適化ツールなども豊富なので，一から開発する場合でもALCHEMYを使用すれば，かなり工数の圧縮が望めるはずである．そしてでき上がったものは，ほかのプラットホームへの移植が容易になるというおまけつきだ．

●「BAROQUE」公式ホームページ
http://www.sting.co.jp/baroque/
●BAROQUEは以下のプラットフォームにて発売されている．
セガサターン(SS)，プレイステーション(PS)，プレイステーション2(PS2)，Wii

画像は製品版(未リリース)のものです．
© STING CO.,LTD

プロが使うツールはここが違う!

3Dグラフィックスを使うアプリ開発

後藤 靖，栗原 希，藤本 文彦

ALCHEMYは，iOSやAndroid以外にもあらゆるプラットホームに対応する商用のゲーム開発環境です．このALCHEMYや，ソーシャルゲームの仕組みを提供できるBENTEN，エフェクトツールのBISHAMONなどのミドルウェアの役割を解説します．

本格的な3Dグラフィックスを使用したアプリの開発

家庭用ゲーム機に見られるような本格的な3Dグラフィックスを扱うゲームをスマートフォン向けに開発する場合，ミドルウェアやゲームエンジンを採用することがあります．

本来，家庭用ゲーム機の開発環境として通常はコンソールベンダーが何らかの開発ツールを提供しており，開発者はそれらを利用して開発できます．しかし，残念ながらアップルやGoogleからは，ゲーム開発に特化したスマートフォン向けの環境を用意していません．すでに自社で内製のゲームエンジンを持っている大手のゲーム会社であれば，それらをスマートフォン向けに対応させて流用していくこともできるかもしれませんが，それ以外の場合は1からゲームエンジンを開発するか既存の製品を利用するということになります．

そこで，スマートフォン向けに3Dグラフィックスを使用したアプリを作れるミドルウェアやゲームエンジンが注目されています．iOSとAndroidの両方に対応している代表的な3Dゲーム開発環境と特徴を表1に示します．

それぞれに特色があり，開発規模，開発スタイルによって選択肢がありー概に何が正解かを判断することは非常に難しいです．ここでは一例としてシリコンスタジオの「ALCHEMY」(図1)を紹介します．また，ALCHEMYと連動してソーシャルゲームを提供できる同社のフレームワーク「BENTEN」，マッチロックの3Dエフェクト・ツール「BISHAMON」を紹介します．

まず，ALCHEMYを使ったiPhone，Androidアプリ開発を解説します．

表1 iOSとAndroidの両方に対応している代表的な3Dゲーム開発環境

名称	特徴
UnrealEngine3	非常に大規模なツールセットを備えた統合型ゲームエンジン． 無料版のUDKでは専用のスクリプト言語のみで開発可能． 有料版はC++で開発可能． 「Infinite Blade」のクオリティはあまりにも有名．
Unity	低価格ながら本格的なツールセットを備えた統合型ゲームエンジン． アプリケーションはC#かJavaScriptで記述する． 近年最も注目されているゲームエンジン．
ALCHEMY	商用販売のみ． iOS，Android以外にもあらゆるプラットホームをサポートするゲーム開発環境． 統合型のゲームエンジンではなくライブラリとツールのセットを提供． C++で開発可能． 10年以上現役で動き続ける優れた基本設計の開発ツール．

図1 ALCHEMYのロゴ

ALCHEMYとは

ALCHEMYは，一言で言えばマルチプラットホームのゲーム開発環境です．現在までに15種類以上のプラットホームでアプリケーションがリリースされた実績があります．

ALCHEMYは，仮想マシンではなく純粋なC++ネイティブ・アプリケーションとして動作します．APIのサポート範囲もグラフィックス機能だけではなく，OSやファイル入出力に関する部分，ゲームパッドやウインドウ制御など非常に広範囲に渡って抽象化されたライブラリとして提供されています．そのため，ソースコードとデータにほぼ100%の互換性があります．これは，ほかのプラットホームの開発環境でソースコードをコンパイル(ビルド)し直すだけで簡単に動作できることを意味します．例えば，Windowsで開発したものを，iOSやAndroidで簡単に動作できるということです．

ALCHEMYのiOS版，Android版のそれぞれの開発環境を表2に示します．ゲームデータの作成環境としてはWindows版が提供されます．Windows上

表2　対応プラットホームと対応バージョン

対応プラットホーム	対応バージョン
iOS	Xcode 4.2
	対応iOS 4.2以上
Android	Android SDK r16以上
	Android NDK r7以上
	対応Android OS 2.3.3以上
	API level 10以上

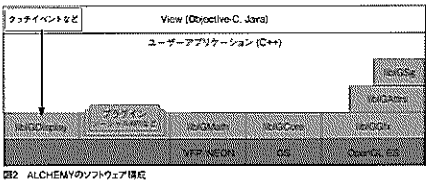


図2　ALCHEMYのソフトウェア構成

でのプロトタイプやDCC(Digital Content Creation)ツールのプラグイン，ツール開発に使用できます．また，モデルビューアや「ALCHEMY Finalizer」といったシーンデータ最適化ツールも提供されます．

ALCHEMYのソフトウェア構成

ALCHEMYを使用したアプリケーションのソフトウェア構成を図2に示します．図には代表的な機能のみ表示しています．

ここではALCHEMYの代表的なライブラリについて紹介します．

・libIGCore：メモリ管理，スレッド管理，ファイルの入出力などOSに関連する機能を抽象化します．

・libIGGfx：テクスチャをセットする，ポリゴンを描画するといった低レベルなグラフィックス機能とそれに伴うリソース管理を抽象化します．スマートフォン版の実装ではOpenGL ES上に構築されますが，ほかのプラットホームではDirectX9やそれぞれのデバイスで用意しているドライバなどの上に構築されます．

・libIGAttrs, libIGSg：シーングラフを抽象化します．

libIGAttrsはlibIGGfxの機能をパッケージ化し，libIGSgはノードツリーなどシーングラフを構成します．シーングラフはキャラクターのアニメーション制御やビュー・フラスタムカリングなど，複雑なシーンの描画に必要な機能を網羅しています．DCCツールから出力されるシーンデータは，このシーングラフを出力します．

・libIGMath：一般的な数学ライブラリを抽象化します．特に高速な処理が要求される演算に関しては，ARMアーキテクチャのVFP/NEONを使用して最適化されています．NEONを持たない場合はVFPまたCPUによる演算に切り替わります．

・libIGDisplay：ウィンドウやキーボード，マウスなどの入出力デバイスを抽象化します．スマートフォン版ではタッチイベントなどをViewから取得してアプリケーションから利用できます．

・プラグイン：プラグインにより，ALCHEMY標準ライブラリ自体を変更することなく一部の機能を実行時に入れ替えたり，拡張オブジェクトを作成したりできます．拡張オブジェクトは，自動的に各ツールに認識されバイナリ・ファイルへの書き込みもサポートされます．

・ソーシャルAPI：プラグインにより主要なソーシャルプラットホームのAPIに対応可能です．

ALCHEMYプログラミング・スタイル

ALCHEMYを感覚的に理解するために，実際のソースコードを紹介しましょう．リスト1のコードは実際に全てのプラットホームで動作します．

ALCHEMYの誕生

当時はパフォーマンスを引き出すのが非常に難しかったPlayStation2向けに，Windowsでも動作するゲーム用マルチプラットホームのライブラリとしてALCHEMYは開発されました．

その後，ゲームキューブやXboxといったゲーム機が次々と発売され，ALCHEMYも順次対応していきました．そのころになると，マルチプラットホーム開発が非常に重要視されるようになり，様々なプロジェクトで利用されました．

過去のプロジェクトとしてPlayStation3，Xbox360，Wii，PSP，Xbox，PlayStation2，Windows PC版のゲームを同じチーム内で同時開発し，同時にリリースした実績があります．

ALCHEMYは，開発当初から10年超える基本設計を念頭に各分野のスペシャリストが開発し，実際そのようになりました．そして現在に至っています．

リスト1　ALCHEMYのプログラミング

```
int main(int argc, char* argv[]) // int iMain(int argc, char*
argv[])
{
    //Alchemy 初期化
    igAlchemy  alchemy;
    Sg::igArkRegisterForIGAFiles();

    //描画領域を確保する(ウィンドウを開く)
    igWindowRef myWindow = igWindow::instantiateRef();
    myWindow->open("Test Window", 640, 480);

    //描画コンテキストを取得
    igVisualContext *myContext = myWindow->getVisualContext();

    //イベントマネージャを作成
    igDefaultInterfaceManagerRef myEvent =
                    igDefaultInterfaceManager::instantiateRef();
    myEvent->addEventProducer(myWindow);

    //フレームバッファをクリアする色を設定
    myContext->setClearColor(igVec4f(0.1f, 0.2f, 0.3f, 1.0f));

    //コンテキストに射影(Projection)マトリックスを設定
    igMatrix44f projMat;
    projMat.makePerspectiveProjection(45.0f, -1.0f, 640.0f/480.0f,
                                      1.0f, 100.0f);
    myContext->setMatrix(IG_GFX_MATRIX_STACK_PROJECTION, projMat);

    //シーングラフの作成(IGBファイルを読む)
    igGroupRef myRoot = makeScene(myContext);

    //シーングラフを処理するトラバーサーを作成
    igCommonTraversalRef myTraversal = igCommonTraversal::instantia
                                                        teRef();
    myTraversal->setVisualContext(myContext);

    //メインループ
    igBool  exitProgram = false;
    while (! exitProgram)
    {
        //トラバーサーにビューマトリックスを設定
        igMatrix44f viewMat;
        viewMat.makeTranslation(igVec3f(0.0f, 0.0f, -5.0f));
        myTraversal->setViewMatrix(&viewMat);

        //シーングラフの生成からトラバースを始める
        myTraversal->reset();
        myTraversal->apply(myRoot);

        //トラバース後のディスプレイリスト(描画リスト)を得る
        igDisplayListAttr *output = myTraversal->getOutput();

        //描画開始
        myContext->beginDraw();

            //フレームバッファ(デプス含)バッファをクリア
            myContext->clearRenderDestination(
                IG_GFX_RENDER_BUFFER_BITS_COLOR |
                IG_GFX_RENDER_BUFFER_BITS_DEPTH);

            //ディスプレイリストを描画
            output->apply(myContext);

        //描画終了(スワップ・バッファリング)
        myContext->endDraw();

        //イベントをチェック・処理する
        myEvent->handle(events);

        if (!myWindow->isOpen())
            exitProgram = true;
    }
    myWindow->close();

    return 0;
}
```

●Windowsなど一般的なプラットホーム
main() // メイン関数
{
　　：
　igAlchemy alchemy;
　　：
}

●iOSやAndroid
iMain() // スマートフォン用メイン関数
{
　　：
　igAlchemy alchemy;
　　：
}

図3　各プラットホームでのmain関数の記述方法

メモリ確保は完全に管理されており，オブジェクトのインスタンスの作成はnewではなく::instantiateRef()を使用することで，メモリ確保場所の指定やメモリ追跡，ランタイムでの機能入れ替えなどさまざまな機能を実現しています．

❸スマートフォン対応

スマートフォン版のALCHEMYで特徴的なところは，C++のみでプログラム開発が完結するところです．一般的にViewと呼ばれるObjective-CやJavaで記述しなければならない箇所は，フレームワークとして提供されます．この部分はソースコードとして提供されるので，必要であれば手を入れることが可能です．

また，マルチプラットホームのAPIなので，Windows版で作成したコードを，そのままスマートフォン版の開発環境でコンパイルし直して動作させることもできます．その場合はmain()関数をスマートフォン版ALCHEMYのエントリポイントであるiMain()に置き換えるだけです(図3)．

Windows版と併用することで，使い慣れたVisual Studio上でプロトタイプやツールを作成できます．それだけではなく，Windows用に作られたソースコードをまったく無駄にせずにスマートフォンやほかのプラットホームに流用することが可能で，さらにターゲットのプラットホームが決まっていないようなプロジェクトでも，とりあえずWindows版から制作を始めることができます．これがマルチプラットホームAPIの強みです．

マルチプラットホームでの開発パイプライン

図4はALCHEMYを使用してマルチプラットホームのアプリケーションを開発する工程を表します．図のように，通常のゲーム開発では，ソースコードを担当するプログラマと，ゲーム内のキャラクターやシーンなどのデータを作成するアーティスト(デザイナー)で作業を分担します．

・プログラマ作業：ALCHEMYのマルチプラットホームAPIを生かして，1

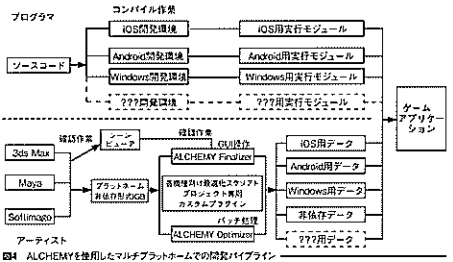


図4 ALCHEMYを採用したマルチプラットホームでの開発パイプライン

図5 ALCHEMY Finalizer

つのソースツリーを管理するだけで完結します．それぞれの開発環境でソースコードをコンパイルし，実行モジュールを作成します．

・アーティスト作業：使い慣れたDCCツールでモデルやアニメーションを作成し，ALCHEMYが提供するエキスポート・プラグインを用いてシーンデータを出力します．

シーンデータには，マテリアル，テクスチャ，アニメーション，カメラ，ライトといった情報をALCHEMY独自のIGBファイル形式で保存します．プラグインは「3ds Max」，「Maya」，「Softimage」といったAutodeskのツールに対応しています．

これらのツールのままでもシーンデータを非依存データとしてアプリケーションに組み込むことが可能ですが，後述する「ALCHEMY Finalizer」を利用してプラットホーム専用の効率的なデータに変換できます．

・新たなプラットホームの追加

マルチプラットホームの強みとして，開発パイプラインを崩さずに新たなプラットホームを追加することも容易になります．図4の「???」の部分は，家庭用ゲーム機であったりアーケードゲーム機であったりします．リリース後の移植作業はもちろんのこと，制作途中であっても柔軟にプラットホームを追加することが可能です．

◎ALCHEMY Finalizer

ALCHEMY 開発パイプラインで重要な役割を持つツールとして，ALCHEMY Finalizer(図5)があります．

ALCHEMY Finalizerは，シーングラフを可視化する機能やノード編集機能も備えていますが，最適化スクリプトによるデータの最適化を行う役割があります．例えば，「シーングラフの階層構造を単純化する」「ジオメトリを三角形ストリップ化する」といったさまざまな処理が利用できます．これらを組み合わせてアプリケーションに最適なデータの作成を一括処理できます．

さらに，プラグインを作成し機能を追加することでゲーム専用のデータを追加したり，特定の名前のマテリアルだけパラメータを修正するなどの制作スタイルに応じた一括処理を組み込んだりできます．このような組み合わせはスクリプトファイルとして保存することが可能で，「ALCHEMY Optimizer」を使用してコマンドラインからバッチ処理を行えます．

データ変換処理を柔軟に構築できるように設計されていると，以下のような利点があります

・デザイナーでなくてもデータ変換：データ出力時にプラットホーム固有のデータが含まれるようなケースでも，デザイナーにデータの再出力を頼む必要がなくなります．

・プラットホームに最適なデータの作成：あるジオメトリが「インデックス付きの三角形」で，高速に処理できるプラットホームもあれば「インデックスなし三角形ストリップ」で高速に処理できるプラットホームもあります．これらに対応したデータの最適化が可能です．

・特化したデータの作成：階層構造の最適化のためにシーングラフを単純化したいけれども，あるノードだけはアプリケーションからアクセスしたいので消さずに残しておきたい場合などそのアプリケーションに特化したデータの作成に利用できます．

ALCHEMYの目指しているところ

図6は，想定されるALCHEMYとUnityのユーザ比較です．携帯アプリ系の

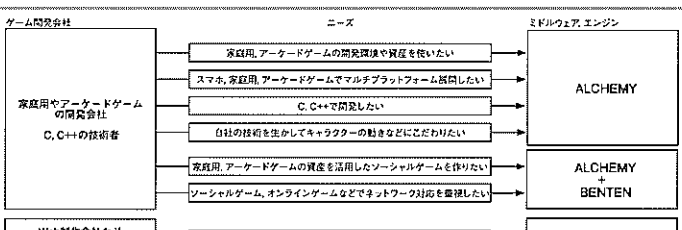


図6 ALCHEMYとUnityの想定ユーザーの比較

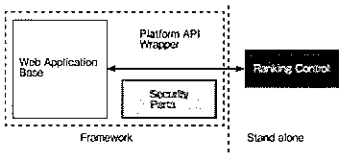


図7 BENTENフレームワーク

開発メーカーと家庭用ゲーム系の開発メーカーでは同じゲームと名の付く業種であっても持ち合わせている技術は大きく異なることを感じています.

現在, さまざまなゲームエンジンがリリースされる中, ALCHEMYの目指しているところは,「家庭用ゲーム機のクオリティ, ノウハウをそのままスマートフォンで利用したい」という開発スタイルに対応することにあります.

家庭用ゲーム機の開発環境は, ほぼ100% Cライブラリとして提供され, 開発者はそれに親しんでいると共に, 膨大な量のC(C++)言語のソースコードの資産を持っています.

資産というのは, 例えばゲーム上のキャラクター同士がぶつかった場合に適度にすれ違うようにしたり, 派手な画面エフェクトの方法であったり, ゲームそのものであったりするプログラム・コードを指します. 資産はいわば開発会社特有のお家芸であり, それを蓄積することでクオリティを上げたり開発工数を減らしたりできます.

ALCHEMYはC++で提供され, もともと家庭用ゲーム機のゲーム開発の現場で利用されてきたので, それらの資産を十分に利用できるようにと考えられています. 実際に過去にPS2向けにALCHEMYを用いて開発されたゲームを1週間足らずでiOS向けにプレイアブルにしたり, アーケードゲームの開発資産を利用してiOS向けにプロモーション用のアプリをリリースしたりした実績もあります.

このように, ALCHEMYは従来の慣れ親しんだ設法でゲーム開発ができるところで, ほかのゲームエンジンとの差別化を図っています.

BENTEN Overview

「BENTEN」(仮称)とはOpenSocial準拠のフレームワーク群です. SNSプラットフォーマー用のポータルサイトからSNSアプリ(Webアプリケーション, ネイティブアプリケーション)のゲーム・サーバまで幅広く対応し, かつ実装を簡易に行うためのフレームワークです. 今回はSNSアプリ部のフレームワークに特化して概要を解説します.

BENTENは, 図7のように大きく四つのブロックに分かれています.

FrameworkはScalaで記述されています. Java仮想マシン(JVM : Java Virtual Machine)上で動作することによりスクリプト言語よりもサーバへの負荷が少なく, サーバのスケールも行いやすくなっています. また, Scalaは強い型付けを持った言語のため単純なエラーは事前に検証を行い, 修正を

リスト2 Web Application BaseでのHello, World

```
class BentenTest extends BentenServlet {
override def consumer_key = "abcdefghijklmn"
override def consumer_secret =
"0123456789abcdefghijklmn"
override def server_url = "www.example.com"
override def application_root = "bentenTest"
override def application_title = "bentenTest"

override def memcached_servers =
Array("localhost:11211")

override def environment = "development"

get("/") {
<html>
<body>
<h1>Hello, world!</h1>
</body>
</html>
```

リスト3 JSONを返すAPIを作成する場合の記述

```
get("/json") {
contentType = "application/json"
"{ \"result\" : \"OK\" }"
}
```

リスト4 LifeCycleEventへの対応

```
class BentenTest extends BentenServlet with LifeCycleEventCatcher
{
```

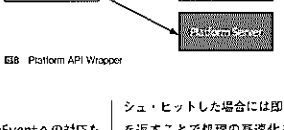


図8 Platform API Wrapper

容易にする利点もあります．さらに，JVMとの親和性もよく，Frameworkを JavaやJRuby, Jython, Quercus(PHP) といったJVM上の他言語からの使用も可能になっています．

Framework部は大きく三つのブロックに分かれており，それぞれの機能を個別に使用することもできます．

❶Web Application Base

「Web Application Base」は，Open SocialのWebアプリケーションの実装を簡易に行うための仕組みです．アプリケーションの実装において，「表示」までに掛かる時間を大幅に改善できます．例えば，"Hello, World" を表示するアプリケーションの場合，リスト2のようなプログラミング実装を行います．この実装では，

・gadget.xmlの自動生成
・OAuth認証の初期化および呼び出し時のシグネチャ検証
・エントリポイントの作成

を自動的に行います．

JSON(JavaScript Object Notation) を返すAPIを作成する場合にもリスト3のように記述することにより，http://www.example.com/bentenTest/jsonというエントリーポイントを作成できます．

また，LifeCycleEventへの対応もリスト4を追加することで，LifeCycle Eventを受信し記録する機能を簡単に追加できます．また，同時にGadget.xmlへ自動的に反映されます．

Web Pageの記述には，以下のテンプレートを使用することもできます．

```
<ul>
#for (i <- 1 to 5)
<li>${i}</li>
#end
</ul>
```

このように，htmlにphpのようなコードを埋め込むことが可能です．テンプレートのフォーマットはScalate (http://scalate.fusesource.org/)対応のものが使用できます．

❷Platform API Wrapper

各プラットホームのAPIを呼び出す際に，ユーザ情報など頻繁に呼び出すものがありますが，1リクエスト中に頻繁に呼び出しを行うとレスポンスの制限時間に引っかかってしまう場合があります．このようなAPI呼び出しをラップすることにより，内部処理で自動的にmemcachedなどのキャッシュサーバへキャッシングを行い，キャッシュ・ヒットした場合には即時に情報を返すことで処理の高速化を図れます．BENTENでは，これを「Platform API Wrapper」(図8)と呼んでいます．

また，この仕組みを利用することにより，呼び出し間隔が制限されているAPIに対しても，制限間隔以内のリクエストはキャッシュを返すことによりサーバエラーを防ぐことができます．この際に，リクエストがキャッシュされたものかの判断を行えます．

APIはScalaのメソッドの形でラップされていて，プログラム中から自然に呼び出せます．People APIの自分自身の情報を取得する場合は，

```
api.People.me.query
```

のような形で行います．戻り値はJSONのScalaでのオブジェクト表現JsObjectクラスのインスタンスに自動的に変換されて返されます．

❸Security parts

「Security parts」は，Webアプリケーションを組む上で必要になるMD5やURL Encode, Base64, OAuthなどを扱うためのユーティリティ群です．一部試験的にPHP互換の関数も実装されています．テンプレートと併用することにより，PHPからの移行を支援します．

図9 Ranking Cached

図10 BISHAMONのロゴ

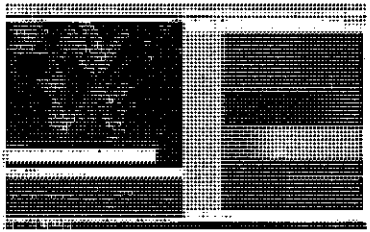


図11 BISHAMONの画面

●Ranking Cached

「Ranking Cached」は，ゲームスコア(得点)のランキングを処理するための仕組みで，独立したプロセスで動くMemcachedに似たメモリ・データベースです．

UserID(整数)とスコア(整数)の組のデータを，常にスコアの降順にソートされた状態で保持します．この時，同点時のランキング処理も考慮されます．例えば，1位のスコアが二人いた場合は次の順位は3位になります．

クライアントの問い合わせに対して，指定ユーザのスコアとランキング，指定順位までのユーザIDリストを返すAPIなどが用意されています．クライアント・プロセスとの通信はTCPで行います．Scala，Ruby，PHPのクライアント・ライブラリが用意されています．

Ranking Cachedはメモリ・データベースのため，データベースの補助としての使用を目的としています．例えば，MySQLと連携させて，スコアのInsert/Update時にトリガーをかけることによってクライアントはMySQL上のテーブルを更新するだけでランキング情報を更新するといったことが可能です(図9)．

また，この仕組みはRanking以外にも応用が可能です．

BENTENのPlatform API Wrapper，Security parts，Ranking Cachedは，iOS/Androidなどネイティブのアプリからの呼び出しも可能です．また，プラットホームを設定ファイルに登録するだけで，各プラットホームに対応した処理を行うため，プラットホームによるAPIの違いを気にすることなく実装が行えます．

スマートフォンとBISHAMON

最近では，特にソーシャルゲームなどのアプリで，グラフィックスの美しさや演出の派手さを売りにし始めています．当然のごとく高品位な映像効果の量産環境が必要となり，アプリ開発需要のラッシュが当面続く中，アプリ開発者自らが簡単操作でエフェクトを作れる要望も多くなっています．それだけでなく，プロのアニメーターの手を借りてコンシューマ向けゲームのようなリッチな演出が求められています．それに対する一つの答えが「BISHAMON」(図10)です．

●BISHAMONとは

BISHAMONは，ゲーム開発現場で生まれた，映像効果作成に特化した3Dアニメーションのツールです．爆発，閃光，自然現象，雷や魔法のような抽象的な表現，ちょっとした3Dオブジェクトをリアルタイムのレンダリングによるプレビューを確認しながら，デザイナーだけで作成することができ，一般的なCGツールで作成するのと比較しても効率よくデータを作成できます(図11)．

従来世代の携帯ゲーム機のような限られたマシン性能の中でも，できるだけコンテンツが派手にリッチに見えるような仕組みにしています．動きには軽量なベクタ・アニメーションを採用しながらも，素材には写真のように精細なラスタ画像(画素の集まりで表現する画像)を用いるハイブリッドなアプローチを採用しています．

また，データ資産が流用できるという強みがあります．大手ゲーム会社だけでなく，演出を重視するパチンコ・パチスロやCGアニメーションの映像素材作りにおいても活躍しています．ほぼすべてのゲーム機やプラットホームで利用されてきたツールです．

アニメーションのデータのサイズが小さく，ネイティブコードとGLESで

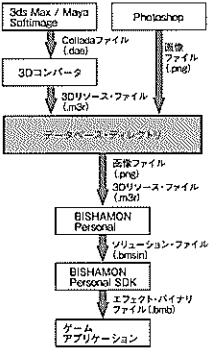


図12　データの流れ

のGPU活用により高速に動作するので，コンシューマ向けゲーム機と同じようにスマートフォンにおいても有効です．ここでは，スマートフォン向けのBHISHAMONについて紹介します．

◎BISHAMONの構成

スマートフォン向けのBISHAMONは二つのツールで構成されています．

一つは，アニメーションのデータ制作ツールである「BISHAMON Personal」です．こちらのツールは単体でも迫真静止画像や動画GIFアニメという形で，CG映像の素材も出力でき，特に商用での制限なく利用できます．

もう一つは，BISHAMON Personalで作成したデータをアプリに読み込んで再生させるための開発環境「BISHAMON Personal SDK for iOS/Android」です．SDKには，エフェクトのデータを各プラットホーム専用に変換するコンバータと，ツール，SDKからなります．

SDKは前年度年商1000万円未満の開発者に利用が限られます．年商1000万円以上の規模の開発者へは，Professional版を用意しています．

これらはマッチロックストア(http://www.matchlock.co.jp/store/)で入手できます．

スマートフォンアプリへの利用方法

BISHAMONをスマートフォンのアプリへ利用する方法を解説します．以下の3ステップで行います．

①ツールでエフェクトを作成する
②①できたエフェクトをiOSやAndroid用にコンバートする
③コンバート済みデータをアプリケーションに組み込む

参考までに，データの流れを図12に示します．Photoshopなどの画像編集ツールで画像ファイルを作成し，データベースのディレクトリに用意します．さらに必要であれば，3ds MaxやMaya，Softimage(|BSOFTIMAG|EXSI)などのDCCツールでモデルを作成し，データベースのディレクトリに用意します．以上のファイルを参照して，BISHAMONでソリューション・ファイルを作成します．ソリューション・ファイルからコンバータでバイナリを作成し，ゲームに組み込みます．

エフェクトを作成するBISHAMON Personalについては，誌面の都合上，詳細は割愛します．マッチロックのホームページ(http://matchlock.co.jp/products)から体験版をダウンロードできます．チュートリアルをこなせば簡単に作成できます．

アプリへの組み込みは，SDKドキュメントとサンプル・プログラムを参考に，プロジェクトにBISHAMONのライブラリを組み込みます．いったん組み込めば，アプリ内でBISHAMONのエフェクトを読み込んで再生することができるようになります．

◎アプリへの組み込み

以下，イメージをつかんでもらうためにBISHAMON Personal SDK for iOSを例に，アプリへの組み込み手順を説明します．Objective-C用に用意されたサンプルを使ってプロジェクトに組み込みます．C++のAPIに直接アクセスして，複雑な制御も行えます．

Bishamon SDKは処理の高速化のために，C++(ネイティブコード)で実装されているので，Objective-CからC++コードを利用するにはソースコードの拡張子を.mmにする必要があります．しかし，いきなりObjective-CからC++のBishamonAPIを呼び出すのは敷居が高いため，Objective-C用の実装サンプルが用意してあります．

実装は，リスト5のように行います．

リスト5　アプリへの組み込み

後藤 靖 Osamu Goto
ソリコンスタジオ株式会社リサーチ&デベロプメント部所属．ALCHEMYの開発，ユーザーサポートを行っています．2001年からALCHEMYの開発とユーザーサポートを担当し，一つでも多くのタイトルに採用していただけるよう日々業務に励んでいます．休日は会社の同僚とトレーニングを楽しんでいます．

```
●LLVM GCC 4.2 - Language
Other C Flags   -I../../include -DBM3_PLATFORM_opengles_glsl
・パスが通っているところにincludeディレクトリを展開した場合は-Iオプションは必要ない
・OpenGL ES1を使用する場合はBM3_PLATFORM_opengles_glslを定義する必要ない

●Linking
●Other Linker Flags
Debug   -L../../lib -lbm3es2d
Release -L../../lib -lbm3es2
・パスが通っているところにlibディレクトリを展開した場合は-Lオプションは必要ない
・OpenGL ES1を使用する場合はlibbm3es1*.libを指定する
```

図13　プロジェクトのビルド設定

リスト6　simpleViewController.hの変更

```
#include "BM3Manager.h" // 追加

@interface simpleViewController :
UIViewController {
@private
 EAGLContext *context;
 GLuint program;

 BOOL animating;
 NSInteger animationFrameInterval;
 CADisplayLink *displayLink;

 BM3Manager *manager; // 追加
 BM3Effect  *effect;  // 追加
}
```

リスト7　simpleViewController.mmの編集

```
●(void)awakeFromNib
{
 // 追加ここから
#ifdef BM3_PLATFORM_opengles_glsl
 EAGLContext *aContext = [[EAGLContext alloc] initWithAPI
:kEAGLRenderingAPIOpenGLES2];
#else
 EAGLContext *aContext = [[EAGLContext alloc] initWithAPI
:kEAGLRenderingAPIOpenGLES1];
#endif
 // 追加ここまで

 // 削除ここから
 EAGLContext *aContext = [[EAGLContext alloc] initWithAPI
:kEAGLRenderingAPIOpenGLES2];

 if(!aContext) {
   aContext = [[EAGLContext alloc] initWithAPI:kEAGLRende
ringAPIOpenGLES1];
 }
 // 削除ここまで

 if (!aContext)
   NSLog(@"Failed to create ES context");
 else if (![EAGLContext setCurrentContext:aContext])
   NSLog(@"Failed to set ES context current");

 self.context = aContext;
 [aContext release];

 [(EAGLView *)self.view setContext:context];
 [(EAGLView *)self.view setFramebuffer];

 if ([context API] == kEAGLRenderingAPIOpenGLES2)
   [self loadShaders];

 animating = FALSE;
 animationFrameInterval = 1;
 self.displayLink = nil;

 manager = [[BM3Manager alloc] init];          // 追加
 effect = [manager createEffect:@"smoke_ks01"]; // 追加

}


●(void)dealloc
{
 [manager releaseEffect:effect]; // 追加
 [manager release];              // 追加

 ...
}


●(void)drawFrame
{
 ...
 // 自動生成された描画処理をコメントアウト
 // glDrawArrays(GL_TRIANGLES_STRIP, 0, 4);

 // 追加ここから
 // 更新処理
 [effect update];
 // 描画処理
 [manager begin];
 [manager drawEffect:effect];
 [manager end];
 // 追加ここまで

 [(EAGLView *)self.view presentFramebuffer];

}
```

大きく分けて四つの処理を行うAPIをアプリのプログラムに書き込むことで完了します．

それでは実際に組み込む例を見ていきましょう．

・プロジェクトの作成

Xcodeを起動して[Create a new Xcode project.]からOpenGL ES Applicationを選択します．Product Nameには任意の名称を記述します．ここでは仮にsimpleとします．

BISHAMON SDKのsample/commonディレクトリごとProjectNavigatorへドラッグ＆ドロップして，ソースコードを追加します．ダイアログでは[Create groups for any added folders]を選択します．sample/dataディレクトリをSupporting Files以下へドラッグ＆ドロップして追加します．ダイアログでは[Create external build system project]のチェックを外します．続いて[Create folder references for any added folders]を選択します．

・拡張子の変更

simpleAppDelegate.mとsimpleViewController.mの拡張子を.mmへ変更します．

・BuildSettingsの設定

プロジェクトのBuild Settingの設定を図13のように行います．

・ソースコードの編集

simpleViewController.hを開き，リスト6のように編集を行います．

栗原 希
Nozomi Kurihara
シリコンスタジオ株式会社オンライン事業部開発マネージャー．システム，サーバ，ネットワークの設計を主に担当しています．[よりシンプルに，ユーザの要求をできる限速にシステムの実現を行っています．

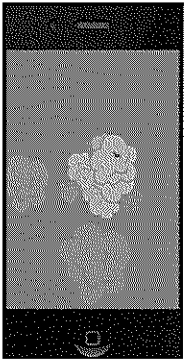


図14 iPhoneで実行

```
BMEffect エフェクトクラス
プロパティ
BOOL expiration      終了したかどうか
float generatingScale      生成倍率
メソッド
-(void)update      更新する
-(void)updateWithFrame:(float)frame      frameフレームで更新する
-(void)reset      リセット(発生し始めに戻す)
-(void)retire      リタイア(消去処理をおこなうように)する
-(void)setMatrix:(el::matrix44 &)matrix      行列を設定する
-(void)setColorScale:(el::color_rgba<float> &)scale      色に倍率をかける
-(void)setColorScaleR:(float)r G:(float)g B:(float)b A:(float)a      色に倍率をかける
-(void)setOffsetPosition:(el::vector3d &)offset位置オフセット
-(void)setOffsetPositionX:(float)x Y:(float)y Z:(float)z      位置オフセット

BMManager マネージクラス
プロパティ
float maxGeneratingScale      生成倍率最大値
メソッド
-(id)init初期化
-(id)initWithVertexBufferSize:(unsigned int)size      頂点バッファサイズ指定付きで初期化
-(void)release      解放
-(BMEffect *)createEffect:(NSString *)name      エフェクト作成
-(BMEffect *)createEffectWithUTF8String:(const char *)name      UTF8指定エフェクト作成
-(void)releaseEffect:(BMEffect *)effect      エフェクトの解放
-(void)drawEffect:(BMEffect *)effect エフェクトの描画
-(void)setView:(const el::matrix44 &)view      ビュー行列の設定
-(void)setProjection:(const el::matrix44 &)projection      射影行列の設定
-(void)begin      描画開始処理
-(void)end      描画終了処理
```

図15 BMのリファレンス

さらに，simpleViewController.mmに編集を行います．リスト7のように初期化処理をawakeFromNibメソッドへ追加し，解放処理をdeallocメソッドの最初に追加．さらに更新処理と描画処理をdrawFrameメソッドの最後の方に追加します．

ビルドすると図14のようにiPhoneにエフェクトが表示されます．

組み込み後は，図15のBMのリファレンスを参考に，エフェクトをプログラムでコントロールできます．

以上のような3Dのコントロールを意識せずに，2D座標で簡単に制御するためのサンプルもあります．

BISHAMONのこれから

BISHAMONは最終的に，スマートフォン時代のユビキタスなインタラクティブ・アニメーション作成ツールとして，デザイナー，アーティスト，ディレクター，両用プロダクトに最も愛される制作環境になることを目指しています．

エフェクトにとどまらず，GUIやインタラクティブなアニメーションのコンテンツ全般を作成できるようにするために，JavaScript制御とイベント・ハンドラが組み込まれたツール「BISHAMON Personal PREMIUM α版」がリリースされます．α版の今なら正規版より安く購入でき，正規版のリリース時に無償アップデートが可能です．

β版以降は，映像効果に強みを置きながらも，キャラクターや2D絵まで含めたアニメーション全般の作成が可能な機能拡張を予定しています．

また，BISHAMONをHTML5形式の出力に対応させると同時に，もとより得意とする各プラットホーム向けのネイティブ対応に関しては，SDKをベースにしたプレイヤーを必要とするプラットホームへ逐度向上のための選択肢として無料配布していく予定です．さらに，たくさんのユーザにBISHAMONのアニメーションのデータを活用してもらえるように，作成したデータのテンプレートを元に，ブラウザ上でお手軽にアニメーションを編集・調整できるツール付きデータ販売サイトの展開も予定しています．

エフェクトやアニメーションに最もこだわりの強い日本から，アニメーションツールのデファクトスタンダードが生まれる必然性を信じて日々開発に取り組んでいます．

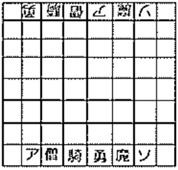
※　※　※

スマートフォン向けゲーム開発の例として，ALCHEMY，BENTEN，BISHAMONの各ツールについて解説しました．この記事を参考にして，今まで家庭用ゲーム機向けに開発してきた開発者の皆さんに，スマートフォンでも高いクオリティのゲーム開発に挑戦していただけたらと思っています．

藤本 文彦
Fumihiko Fujimoto
マッチロック株式会社代表取締役．ゲーム開発会社がリジンマジックでメインプログラマ，ディレクター，経営役員を経めた後，日本のクリエイターが世界でより活躍できるツールを作るための会社を作りました．今は世界のモバイルコンテンツ市場と日本の優秀なアニメーターをBISHAMONでつなげることが目標です．

Appendix2
ゲーム開発者になるためにしておくべきこと

藤本 文彦



図A 問答無用戦記

ゲーム開発者になりたい，という声をよく聞きます．そのために必要なことを考えてみましょう．

◎ゲームを1本作ること

まずはどんな規模であれ，自分一人で1本のゲームを完成させることです．そして，その自分のフレームワークを常に心の中に一つ持ちながら，新しい知識で肉付けしたり，アップデートしていくことです．ゲームを実際に作れるようになるには，ゲーム会社で何年も下積みをすれば作れるようになるわけではありません．ゲームを「えいや」と作り始めて完成させる以外に，ハードルを下げる方法はないと思います．

今すぐゲームを1本完成させられないとしたら，それは，ゲームのレシピに自分で分解もできないような巨大なものをいきなり作ろうとしているのが原因かもしれません．レシピが思いつく範囲の小さなゲームを考えて，とにかく作り始めましょう．

◎他人を楽しませること

今持っている材料(技術や知識)にちょっとプラスをすれば作れる程度のゲームが思い浮かばないこともあるでしょう．作ろうというモチベーションが湧くアイデアが出てこないというのは，自分が決めた仕組みやルールによる遊びで，人を巻き込んで楽しませたという原体験がないからかもしれません．しかも，いきなりプログラミングの習得などの地味で巨大なハードルを越えようとしてもモチベーションは上がらないと思います．

手持ちの技術や知識だけで，他人を楽しませる仕組みやルールを作りあげる体験をたくさん積みましょう．C言語がかろうじて組めるようになったけど，printfしか表現力がありませんというのであれば，バトルの計算だけ肩代わりしてくれるプログラムを使ってもいいです．何もなければサイコロと鉛筆でもいいのです．トランプを利用した新しいゲームを考案してもいいです．そしてすごく簡単なゲームを作って，友達に付き合ってもらいましょう．職場や学校の休み時間に数プレイできるくらいのものがいいです．このあたりは，ソーシャルアプリに親和性のある，複雑すぎないゲーム性を考えられる引き出しを持っているかにもつながります．

本当にその遊びで盛り上がれる経験ができたのなら，だんだん，より複雑なオリジナルのルールの遊びを考えだすようになっていくでしょう．そうやってゲームシステム脳がだんだんとできていくのだと私は思っています．そんな中でゲーム制作界隈の情報を収集していれば，いずれプログラムやゲーム開発環境という武器が欲しくなるでしょうし，その武器を利用してゲームを作るモチベーションができるはずです．

◎とにかく遊ぶこと

もし，小規模なシステムで盛り上がるような遊びを考えることができないなら，それはそういったゲームや遊びで盛り上がった経験が足りないからです．大規模で複雑なルールのビデオゲーム以外にも，パーティゲームとか，ボードゲーム，カードゲームやカジュアルゲームとか，ルールの複雑度の低いゲームをいくつも体験して，楽しいと思ったら，自分でシステムを組むことができるレベルの遊びの引き出しも増えていくでしょう．

筆者の例でいうと，パソコンのプログラムを一部利用したボードゲームを作りました．対戦型のシミュレーションRPGで「問答無用戦記」とかいうカッコ悪いタイトルでした(図A)．タイトル通り，シナリオも何もなく，将棋のように対戦がいきなり始まります．これはキャラクターが1対1で対戦するゲームより，友達の友達くらいの広い範囲で長期間流行りました．友達もいやいや付き合ってくれている感じではなく，トーナメントも行われ，とても盛り上がりました．ハイコンテクストなゲームではありましたが，市販ゲームソフトやカードゲームに劣らない遊びを作り上げた瞬間でした．この原体験は，今の自分にも色濃く影響していると思います．その後，調子にのって，より複雑なゲームをいくつも作っていくわけですが，それらは複雑すぎて流行りませんでした．

あとは，考え方の選択肢を増やすためにも，ディジタルとアナログの入り混じったような独創的なゲームでたくさん遊んでおくといいかもしれません．私は，ファミコン版の源平討魔伝(ボードゲームが付属している！)や，バーコードバトラーが印象に残っています．

ディジタルとアナログを混在させると，自分が作れない複雑なアルゴリズムや表現を，ほかのもので肩代わりさせるというアイデアの引き出しができ上がります．

◎モチベーションの源を

こういった体験が，ゲームの完成形や現実的な落とし所に多様な選択肢をもたせてくれたり，モチベーションの源になったりし，挫折せずに少しずつビデオゲームのような複雑なシステムを作っていける土台になるのだと思います．