

目次

はじめに	2
1 Emoji	11
第1章 Emoji とは	12
1.1 Emoji の定義	12
1.2 Emoji の効果	12
1.3 Emoji の欠点	13
第2章 Emojicode とは	14
2.1 概要	14
2.2 導入と使用	14
2.3 文法	15
標準入出力	16
数値演算	17
文字と文字列	18
変数と定数	18
真偽値と制御構文	19
リスト, 辞書	20
無名関数	21
クラス	22
2.4 利点と欠点	23
第3章 Emojicode を用いた制作例	24
第4章 結び	25
付録 / Emoji 対応表	26

2	ハッシュライフの話	28
第1章	ハッシュライフとは	29
第2章	計算方法	30
2.1	ノードの計算	30
2.2	葉の計算	34
2.3	木の拡張	34
第3章	例	36
第4章	2^n 世代後の計算方法	38
第5章	メモ化	40
第6章	省メモリ化	41
第7章	まとめ	42
第8章	参考	43
3	MonoGame でクロスプラットフォーム開発	44
第1章	公式ドキュメント	45
第2章	環境構築	46
2.1	Visual Studio, Xamarin	46
2.2	MonoGame	46
2.3	Android エミュレータの設定	47
2.4	Android 実機デバッグの設定	47
	Android の USB デバッグの設定	47
	Visual Studio 等の設定	47
第3章	開発環境上でのコンパイル・実行	48
第4章	Android でのコンパイル・実行	50
第5章	ソースコードの共通化	53
5.1	TestApp Solution に SharedProject を追加	53
5.2	Game1.cs を SharedProject に追加	54
5.3	TestApp-Windows/TestApp-Android プロジェクトの Game1.cs を削除	55
5.4	Windows/Android プロジェクトの Reference に SharedProject を追加	55
5.5	依存関係の調整	55

第 6 章	画像の表示	56
第 7 章	リソースの共通化	59
第 8 章	画像サイズの調整	62
8.1	画面サイズの取得	63
8.2	画面サイズに合わせた画像の伸縮	63
8.3	画面のフルスクリーン化	63
第 9 章	入力の取得	64
9.1	Input.cs	64
9.2	Android の入力	65
9.3	Windows の入力	65
第 10 章	残りの STG を作る	66
第 11 章	まとめ	67
第 12 章	おまけ	68
12.1	テキストの描画	68
12.2	Android のタッチ入力について	68
12.3	MonoGame の有効な使い方	69
12.4	現在分かっている不便なこと	70
	テキストファイルの読み込み	70
	関数のオーバーロード	70
4	yield(Generator) の仕組みを探ってみた	71
第 1 章	まえがき	72
第 2 章	C# の場合	73
2.1	まとめ	78
第 3 章	ChinoScript の場合	79
3.1	まとめ	81
5	D 言語はいいぞ	82
第 1 章	D 言語って？	83
1.1	コンパイラが いいぞ	83
1.2	開発環境が いいぞ	83

1.3	言語仕様がいいぞ	84
1.4	テンプレートがいいぞ	86
1.5	マスコットがいいぞ	90
第 2 章	いいぞ...?	91
第 3 章	D 言語はいいぞ	92
6	Bayes 学習の理論と特異 Bayes 情報量規準 sBIC	93
第 1 章	序文	94
第 2 章	理論	96
2.1	Bayes 推測の枠組み	96
	事後分布	97
	Bayes/最尤の予測分布	97
	重要な確率変数	99
2.2	統計的漸近理論	101
	昔話	102
	漸近理論	102
	学習の状態方程式	104
2.3	代数幾何と Bayes 学習理論	106
	前書き	106
	特異点解消定理とゼータ函数	107
	学習係数としての実対数閾値	109
	具体的なモデルの実対数閾値	111
2.4	理論研究の意義	115
第 3 章	情報量規準	117
3.1	AIC と WAIC	117
	赤池情報量規準 AIC	117
	広く使える情報量規準 WAIC	119
	弱点	120
3.2	BIC と WBIC	120
	Bayes 情報量規準 BIC	120
	広く使える Bayes 情報量規準 WBIC	121
	弱点	122
3.3	特異 Bayes 情報量規準 sBIC	123
	新兵器参上	123
	定義式……の前に	123

モデル選択問題	126
sBIC の定義式=計算アルゴリズム	127
考察	131
第 4 章 総括	132
4.1 理論	132
4.2 情報量規準	132
第 5 章 参考文献	134
5.1 書籍	134
5.2 論文	135
付録 A / 統計モデルに課される仮定	136
仮定のある統計理論に意味はあるか	136
逆問題	136
統計モデルが満たす仮定	137
付録 B / 本記事筆者おすすめの『ベイズ統計の理論と方法』の読み方	138
統計学になじみ, それから理論を理解する	138
なじみフェーズ	138
理論を理解する	139
付録 C / 演習と解答	142
演習問題	142
解答例	142
7 ガチャキング制作記 (サーバー編)	143
第 1 章 課金王者ガチャキングとは?	144
開発チームメンバー	145
第 2 章 製作環境について	146
2.1 traQ	146
2.2 Gitea	147
2.3 crowi	147
2.4 OwnCloud	147
2.5 HackMD	147
2.6 Discord	147
第 3 章 チーム決定からコーディング開始まで	148
10 月 18 日	149

10月21日	149
11月15日	150
12月5日	151
第4章 コーディング開始	154
4.1 12月6日～8日	154
4.2 12月10日～12月31日	154
4.3 1月1日～	155
1月4日	156
1月5日	157
1月6日	157
1月15日	157
1月16日	158
1月17日	158
1月20日	159
1月22日	159
2月15日	159
2月18日	159
2月27日	159
3月2日	160
3月6日	160
3月8日～13日	161
3月18日	162
3月22日	163
3月25日～4月5日	163
4月6日	163
第5章 完成して	164
5.1 完成した要因	165
サーバー・クライアント担当の密な連絡	165
それぞれがしっかりとした意見を持っていた	165
ゲームを遊びながら開発を進められた	165
Javascript という言語特性	165
5.2 最後に	165
8 ガチャキング制作記 (クライアント編)	166
第1章 開発の経緯	167
1.1 プロジェクト開始まで	167

1.2	雰囲気	168
第 2 章	言語とライブラリ	169
2.1	JavaScript に至るまで	169
2.2	完成したライブラリ	170
第 3 章	全体構造	171
第 4 章	プログラムの細かい話	173
4.1	アニメーション / Easing	173
4.2	DOM 要素との協調動作	174
4.3	View とマウス	175
4.4	ボタンの設計	177
4.5	ES6 → JavaScript	177
第 5 章	ゲームに関する話	178
5.1	真ん中のフィールド	178
5.2	選択 UI	180
	数字について	181
5.3	妙に面白いことが起きる	181
第 6 章	開発に関して	182
6.1	チーム開発	182
	情報伝達 / traQ	182
	集会 / 諸サービス	184
6.2	まだ β 版	184
6.3	プログラムのあり方	185
	リソースとしてのプログラム	185
	メッセージパッシング	186
第 7 章	終わりに	187
9	並行性への道	188
第 1 章	継続の力	189
1.1	コールバック	189
1.2	継続の連鎖	190
1.3	型	190
1.4	一般化	191
1.5	通常の処理を継続に載せる	191
1.6	call/cc	193

1.7	アロー関数とジェネレータ	194
1.8	まとめ	196
第 2 章	並行プロセス	197
2.1	継続の戻り値	197
2.2	命令とスケジューラ	198
2.3	まとめ	199
第 3 章	時間停止と共有変数	200
3.1	実行の遅延	200
3.2	Box	201
3.3	時間待機	203
3.4	まとめ	203
第 4 章	チャンネルとプロセス	204
4.1	spawn	204
4.2	補足：コンテキストスイッチ	205
4.3	まとめ	205
第 5 章	早い者勝ち	206
5.1	EventListener	206
5.2	必要な概念	207
5.3	Box と check 関数	207
5.4	Cont の構造	208
5.5	joinAny	209
5.6	onTerminate	209
5.7	おまけ：joinAll	210
5.8	おまけ 2：実用例	211
第 6 章	全ての導出	212
6.1	Cont Monad	212
6.2	Concurrency と Box と spawn	212
6.3	$H \rightarrow ((A, H) \rightarrow R) \rightarrow R$	213
6.4	補足	213
	著者紹介	214

Emoji

Double_oxygeN 著

第 1 章で Emoji の定義，第 2 章では Emoji を使うプログラミング言語「Emojicode」の紹介，第 3 章で実際に Emojicode を用いた制作を行う。付録として，Emoji とその Markdown での表記法，Emojicode 上での意味をまとめる。

第 2 章

Emojicode とは

さて、第 1 章で紹介した絵文字であるが、これを文法に取り込んでしまったプログラミング言語がいくつかある*¹。その一つが「Emojicode」だ。公式ホームページはこちら。

<http://www.emojicode.org/>

2.1 概要


Emojicode は、オーストリア在住の Theo Weidmann 氏により開発が進められているプログラミング言語の 1 つである。

基本的にはオブジェクト指向の言語であるが、第一級オブジェクトとして関数も扱えるなど関数型言語としての側面も併せもつ。静的型付け言語で基本的に型推論が有効であり、コンパイルして Emojicode 独自の仮想マシン (**Emojicode Real-Time Engine** と呼ばれる) 上で動かすことができる。他にも現代的な機能を多く搭載しており、見た目とは裏腹に本格的なプログラミング言語となっている。

2.2 導入と使用


使用するだけなら、Try It Online(<https://tio.run/#emojicode>) というオンラインサービスでできる。

導入するには、Magic Installation という方法がある。

2017/7/18 に、バージョンが **0.5** に更新された。(記事執筆中での出来事で大変驚いている。) Mac, Linux の場合は下記 URL の「 Magic Installation」に従ってコマンドを実行すると、インストールが完了する。<http://www.emojicode.org/docs/guides/install.html>

Windows の場合は、Bash on Ubuntu on Windows を利用することで導入できるらしい。

さて、これから実際にコーディングをしていくのだが、Emoji が入力できないとプログラムは一向に完成しない。前述の通り、Emoji の入力は多くの場合簡単ではない。文字コード表から地道

*¹ その他にも、Emojilisp(<http://emojilisp.com/>) や  ++(<https://github.com/rrshaban/keyplusplus>), さらに Emoji(<https://esolangs.org/wiki/Emoji>) といった言語がある。しかし、これらの言語は一般に難解プログラミング言語と呼ばれるものに分類されており、紹介する Emojicode に比べて実用性に乏しい。

ハッシュライフの話

kriw 著

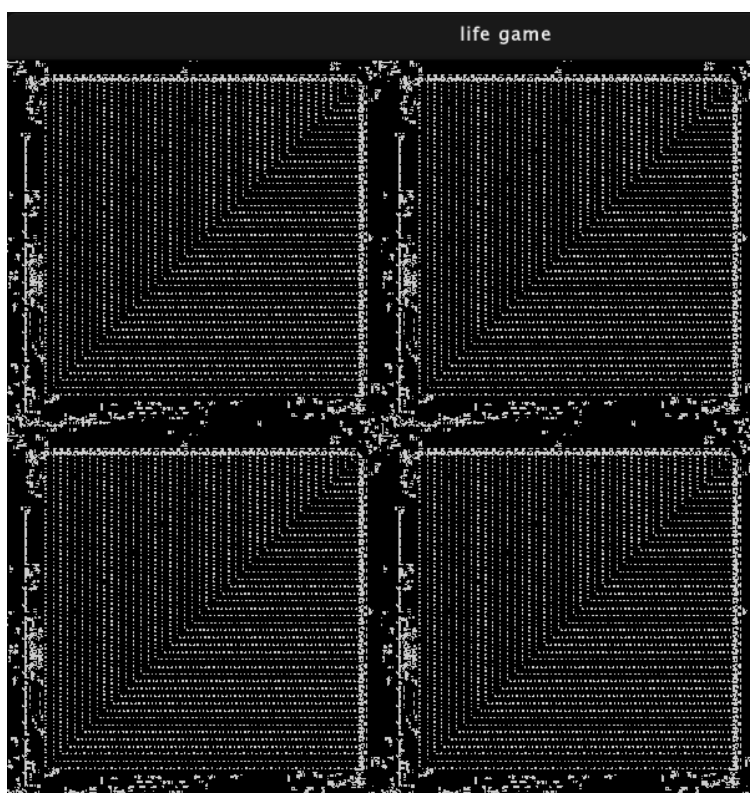
ハッシュライフとは、ライフゲームの計算方法の一つです。ライフゲームを計算する時、単純な実装では各セルを順番に見て行ってそのセルと周囲の生きてるセルをカウント、世代の更新。という風になります。また、セルの状態は2通り（1bit）なので変数に複数のセルを格納して一辺に計算することも出来ます。（ビットボードとか言われてる。）もちろん、GPU を用いて計算することもでき、CPU では敵わない速度で計算出来てしまいます。でも、CPU だけで高速に大きなフィールドでライフゲームを遊びたい。。。そこでハッシュライフです。

第1章

ハッシュライフとは

ハッシュライフとはセルが生きているフィールドを四分の一に分割していき四分木を再帰的に構築してあげます。領域を小さく分割するとそれほど多くないパターンが出現します。これらのフィールドの計算結果を記憶するとそこそこ早く計算が出来るようになるという話です。あと、計算を工夫すると2の冪乗世代後を一度の計算ルーチンで計算できるようにもなります。

こちらがハッシュライフでの計算例です。そこそこ大きいフィールドの計算も可能になります。



<https://github.com/kriw/hashlife> にコードがあります。

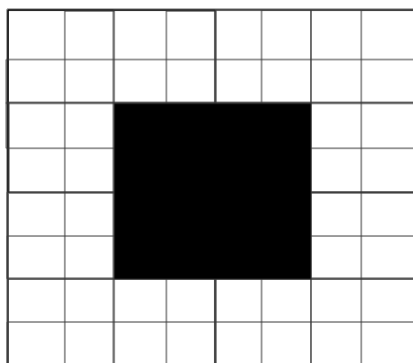
第2章

計算方法

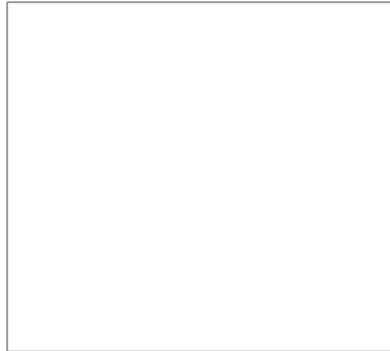
ライフゲームのフィールドを四分木で分割して計算します。このときの木の末端を葉、そうでない部分をノードと呼ぶことにします。

2.1 ノードの計算

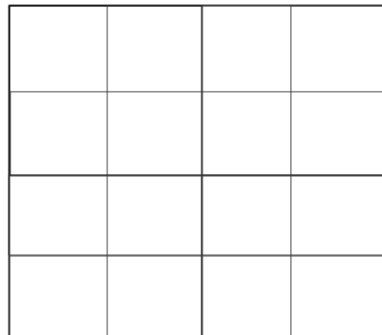
まず、フィールドを用意します。前提として、この計算方法で更新される領域は以下の黒い部分です。



塗りつぶされた領域を更新するために、まずは木を分割しましょう。



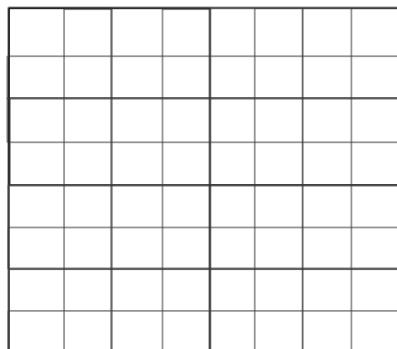
これを 16 分割します。高さ 3 の四分木が出来ました。



また、Node クラスは次のように書き、Node は 4 つのノードを含みます。

```
class Node {
    //左上、右上、左下、右下
    Node nw, ne, sw, se;
    /*
    その他のメンバ変数、関数は省略
    */
}
```

先程の四分木をもっと分割します。



MonoGame で クロスプラットフォーム開発

Namazu 著

MonoGame はクロスプラットフォーム開発が出来るゲームエンジンで、Win/Mac/Android/iOS など多くの OS に対応しています。使用言語は C# です。

Xamarin によって API が提供されており、必要とあらば直接 Native の API を叩くことが出来るのが特徴です。私の開発 OS は Windows ですが、Visual Studio for Mac (<https://www.visualstudio.com/vs/visual-studio-mac/>) が出ていますので Mac での開発も可能です。Linux は Xamarin のサポート外ですので、非推奨です。

なお、本記事では一通りの環境を構築した後に Windows と Android で動く簡単な STG モドキを作ることを目標にしています。STG のソースコードは以下にあります。

<https://github.com/BlondeNamazu/MonoGameSTG>

第 2 章

環境構築

まず、必須の開発環境は

- Visual Studio
- Xamarin
- MonoGame

です。

また、私は Android 上で実機デバッグを行っていますがエミュレータで実行する方法もあります。それぞれ後述しますので、皆さんに合った方を選んでください。私は iPhone 端末を持っていないので申し訳ございませんが iOS については分かりかねます。大筋は同じですが、細かい部分で差異が出てくるかと思われます。

2.1 Visual Studio,Xamarin

Microsoft が Xamarin を買収してくれたので、これらのインストールが一緒にできます。以下のページに Visual Studio2017 を Xamarin とともにインストールする手順が丁寧に記述されていますので、そちらをご覧ください。Visual Studio は持つてゐるけど Xamarin は持つていない…という方向けにも解説されています。

Xamarin やりたい人向け Visual Studio2017 インストール手引書 :

<http://ytabuchi.hatenablog.com/entry/visualstudio2017>

Visual Studio のエディションは Community で大丈夫です。Xamarin も無料なのでご安心ください。

2.2 MonoGame

MonoGame は以下のリンクからダウンロードできます。

MonoGame 3.6 Downloads :

<http://www.monogame.net/2017/03/01/monogame-3-6/>

MonoGame 3.6 for Visual Studio をインストールしてください。

2.3 Android エミュレータの設定

こちらは Visual Studio を新しくインストールした場合デフォルトで入っています。新しくインストールする場合は AVD(Android Emulator Manager) からエミュレーターを作成してください。デバイスは好きなものを利用させていただいて構いません。OS は 6.0 以上がいいでしょう。

2.4 Android 実機デバッグの設定

少々面倒ですが、一旦設定を終えてしまえばこちらの方が断然楽ですしテストもしやすいので私はこちらを推奨します。

Android の USB デバッグの設定

まず、Android の開発者モードを有効にします。この設定をしたことが無い方は「(お持ちの OS のバージョン) + 開発者モード」でググってみてください。大体の場合、設定→端末情報などからビルド番号の項目を連続でタップし続けていると開発者モードが有効になります。そのうえで設定→開発者向けオプションから USB デバッグを有効にします。これで Android 側の USB デバッグの設定は完了です。

Visual Studio 等の設定

これについては、スクリーンショット付きで説明されているこちらのサイトが分かりやすいのでご覧ください。

Windows10 × Visual Studio 2015 で Android 実機デバッグ (Xamarin) :

<http://shunsukekawai.hatenablog.com/entry/2015/10/13/184522>

こちらのサイトの「PC 側での接続確認」が正しく出来れば大丈夫です。なお、adb コマンドは Visual Studio のツールバーから開ける Android ADB Command Prompt で叩くと安心です。

上記二項目が出来れば環境構築は終了です。お疲れ様でした。

第3章

開発環境上でのコンパイル・実行

さて、早速ですがプロジェクトを作っていきます。Visual Studio を起動して、File->New->Project を選択します。新しいウィンドウが開きますので、名前を以下のように設定して MonoGame Windows Project を選択し OK を押します。

Project Name : TestApp-Windows

Solution Name : TestApp

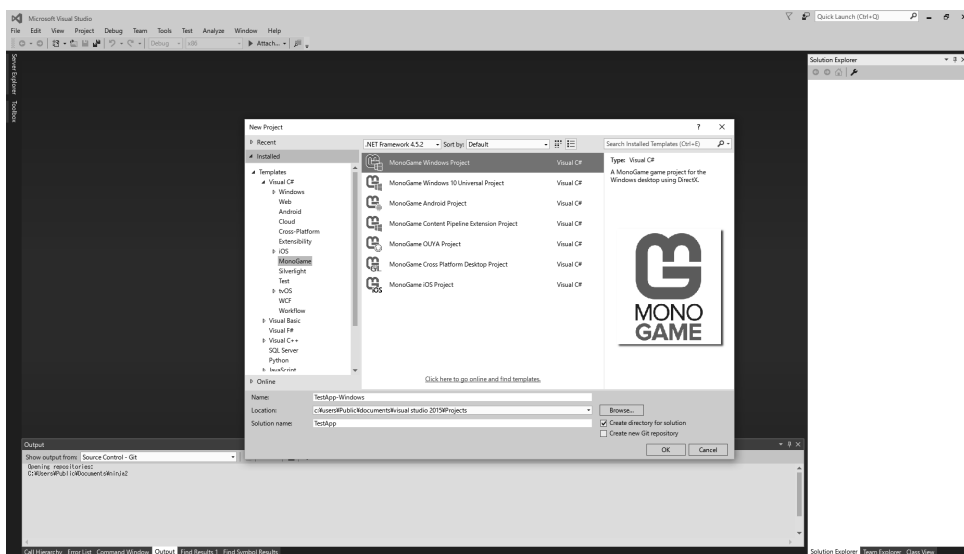


図 3.1: 新規ウィンドウ：Solution・Project の作成

すると色々なファイルが追加されたことが右の Solution Explorer を見ると分かります。この状態で既に画面を表示することが出来ますので、ツールバーの Start を押して実行してみましょう。

yield(Generator) の 仕組みを探ってみた long_long_float 著

こんにちは、Vol.1 で LLVM の記事を書いていた人です。最近はあるゲームにはまっています*1。

今回は表題の yield の内部の仕組みを調べて、自分の処理系に組み込んでみるところまで解説します。yield を知らない人のために説明すると、Generator という値を 1 つずつ返していく関数で、その値を返すための構文です。

*1 ランク 100 弱の新米王子です

第 1 章

まえがき

今回は表題の yield の内部の仕組みを調べて、自分の処理系に組み込んでみるところまで解説します。yield を知らない人のために説明すると、Generator という値を 1 つずつ返していく関数で、その値を返すための構文です。文章よりコードを見たほうが早い気がするので以下のコードで説明すると、

コード 1.1: Generator のサンプル (C#)

```
using System;
using System.Collections.Generic;

class Code1 {
    static IEnumerable<int> f() {
        yield return 1; // ここで一旦呼び出し元に処理が返る
        yield return 2; // 次の呼びだされた時はここから
        yield return 3; // 上に同じ
        // ここまで来たら終わり(MoveNextがfalseを返す)
    }

    public static void Main(string[] args) {
        var gen = f().GetEnumerator();
        var hasNext = false;

        hasNext = gen.MoveNext(); // fの次を取る(≒fを呼ぶ)
        Console.WriteLine(gen.Current);
        hasNext = gen.MoveNext();
        Console.WriteLine(gen.Current);
        hasNext = gen.MoveNext();
        Console.WriteLine(gen.Current);

        Console.WriteLine(hasNext); // => True
        hasNext = gen.MoveNext(); // もう次がないのでfalse
        Console.WriteLine(hasNext);
    }
}
```

うーん、伝わったでしょうか。なかなか説明が難しい気がするので実際に手元で動かしてみることをお勧めします。とにかく、yield は通常の手続き的な処理から逸脱しているすごいヤツなのです*1。これを備えている言語として C#, JavaScript(ES6), Python, PHP*2などが挙げられます。Ruby にも yield はありますが別物です(考え方は似ていますが)。yield がない言語でも継続やスレッドを用いて実現できますが、今回は言語に組み込まれている場合を解説します。

*1 これ自体の考え方は継続やコルーチンなど昔からありましたが、ジェネレータ・yield という形で言語に組み込まれたのは比較的最近のような気がします

*2 これは割と驚きました

第2章

C#の場合

では早速仕組みを探っていこうかと思います。仕組みを探るのに生成物や処理系自体を読むというアプローチを取ります。調べる対象として先ほどのコードを少し修正した以下を使います。

コード 2.1: yield1.cs

```
using System;
using System.Collections.Generic;

class Yield1 {
    static IEnumerable<int> f() {
        yield return 1;
        yield return 2;
        yield return 3;
    }

    public static void Main(string[] args) {
        var gen = f().GetEnumerator();

        gen.MoveNext();
        gen.MoveNext();
        gen.MoveNext();
    }
}
```

また、使用する環境 (ツール) は以下の通りです

- Ubuntu 16.04
- mcs: 5.0.1.1 (C# 6 相当)
- monodis: バージョン不明 (恐らく上と同じ)

まずコンパイルしたものを monodis を用いて逆アセンブルしてみます。

```
$ mcs yield1.cs
$ monodis yield1.exe
```

D 言語はいいぞ そばや著

ここでは D 言語というプログラミング言語を紹介していきたいと思う。主に興味を持ってもらうのが目的なので、実用性よりは面白い機能を紹介するほうに重点を置きたいと思う。願わくばあなたがめくるめく D 言語 er としての一步を踏みだしてくれますように。

第 1 章

D 言語って？

D 言語とは 2001 年に登場した C 言語ベースのマルチパラダイム言語である。現在も活発に開発が行われており、先日 GCC に入った。「Quantum Break」という PS4 のゲームの敵 AI 部分に組み込まれるなど商業利用の実績もある。

1.1 コンパイラがいいぞ

D 言語のコンパイラは 3 つ存在する。1 つは公式が出している DMD(Digital Mars D compiler)。アップデートが盛んで、新機能がいち早く乗るのはこのコンパイラだ。次に、GDC(GCC-based D Compiler)。その名の通り GCC をバックエンドに使うコンパイラだ。最適化性能が高く、実行速度も C++ に引けを取らない。GCC 入りを果たせたのはこのコンパイラのおかげだ。最後に、LDC(LLVM-based D Compiler)。これもその名の通り、LLVM を吐くコンパイラだ。iOS や Android にも (開発中だが) 対応している。いずれも公式サイトからダウンロード可能だ。ちなみにこの記事は DMD2.074.1 で書いている。

1.2 開発環境がいいぞ

D 言語のツールはいくつか作られている。代表的なものとしては DUB というツールだ。いわゆるパッケージマネージャで、世界各国に散らばる D 言語 er 達のコードがこれを使えば落としこける。有名なパッケージとしては Web フレームワーク `vibe.d` や有名な C 言語のライブラリのラッパー `derelect` などがある。開発環境としては Visual Studio の D 言語用プラグイン「Visual D」や Eclipse のプラグイン「DDT」、VSCode のプラグイン「code-d」などがある。ただし Eclipse のプラグイン DDT は開発が停止しているため事実上使うとしたら Visual D か code-d のどちらかがいいだろう。どちらも一長一短があるが、私の意見としてはデバッガが使いたければ Visual D、そうでなければ code-d がおすすめだ。(とはいいつつ私はどちらも使わず Vim で開発している)

1.3 言語仕様がいいぞ

「マルチパラダイム」と謳ってはいるが、D 言語は基本的に手続き型の言語仕様を持ち、標準ライブラリで関数型 (でよく使われる関数) がサポートされているというのが実態である。以下にいくつかテストプログラムを書いてみる。

```
1 import std.stdio;
2 void main() {
3     writeln("D 言語はいいぞ"); // D 言語はいいぞ と表示される。
4 }
```

他の手続き型の一般的な言語と同様、main 関数を宣言するとそこからプログラムが始動する。外部モジュールは import 文で読み込まれ、C++ のように namespace 宣言を別途で行う必要はなく、接頭辞なしでモジュール内の関数が実行できるようになる。

```
1 import std.stdio;
2
3 int getThree() {
4     return 3;
5 }
6
7 void getThreePointer(ref int x) {
8     x = 3;
9 }
10
11 void main() {
12     writeln(getThree()); //3 と表示される。
13     int a;
14     getThreePointer(a);
15     writeln(a); //3 と表示される。
16 }
```

関数の宣言、および使い方も C や Java などと変わらない。D 言語は C 言語のライブラリを読み込む機能を有しており、C 言語との互換性を保つための機能がいくつか存在する。そのため C 言語のポインタもサポートされているが、いわゆるポインタ返しのようなことをしたい場合は C# のように関数の引数に ref や out といったキーワードを充てることで対処するのが望ましい。

```
1 import std.stdio;
2 class Calculator {
3     int x;
4
5     this(int x) {
6         this.x = x;
7     }
8     void add(int y) {
9         x += y;
10    }
11 }
12
13 void main() {
14     Calculator calc = new Calculator(3);
15     calc.add(2);
16     writeln(calc.x); //5 と表示される。
17 }
```

クラスも宣言できる。コンストラクタを this キーワードで宣言するという点以外は普通だ。D 言語は GC があるため、明示的な解放は必要ない。(というかできない) また、ここでは省略するが構造体や共用体やインターフェースも存在する。

Bayes 学習の理論と 特異 Bayes 情報量規準 sBIC Keijan 著

機械学習でよく用いられる統計モデルの多くは階層的な構造や隠れ変数を持ち、パラメータの集合から確率密度関数の集合への写像が 1 対 1 ではないため統計的な正則性を持たない。このようなモデルにおいては、尤度関数や事後分布は Gaussian で近似することができず、従って AIC や BIC といった従来の正則性を仮定する情報量規準は使用できない。本記事では 2017 年に新たに発表された、非正則なモデルにも適用可能な情報量規準である特異 Bayes 情報量規準 (sBIC) について紹介する。情報量規準の背景となる Bayes 統計の理論について概説し、後半でいくつかの情報量規準と共に sBIC を紹介する。

第 1 章

序文

神経回路網，混合正規分布，縮小ランク回帰，隠れ Markov モデルといった階層的な構造や隠れ変数を持つ統計モデルは機械学習・データサイエンス・パターン認識で広く利用されている。統計学の本領はサンプルから計算できる量を用いて推測・学習結果を数学的かつ定量的に評価することであり，統計的推測・学習において性能評価を通してモデルの設計を行うモデル選択に直面する。機械学習の推測についても統計的推測と同様なため同じ問題に立ち向かうことになる。モデル選択の指標としては汎化損失（予測損失）を小さくする或いは周辺尤度を大きくするというものが挙げられ，これらの意味で適切なモデルを選ぶことが求められる。このような問題はモデル選択問題と呼ばれ，古くは負の対数周辺尤度（統計力学における自由エネルギー）を最小化する手法が提案されるなど，研究が行われてきた。自由エネルギーの計算には膨大な Riemann 和が必要となり，汎化損失はデータの生成元である真の分布がわからなければ計算できない。そのためそれらの近似として Bayes 情報量規準 BIC や赤池情報量規準 AIC が考案された。しかしどちらの規準もパラメータの集合から確率密度関数の集合への写像に単射性が仮定されている。このような統計モデルは統計的正則モデルと呼ばれ，正則でないモデルは統計的特異モデルと呼ばれる。本段落冒頭で述べたような階層構造や隠れ変数を持つモデルは特異モデルであり，特異モデルは特殊なモデルではなく広く実用の場で用いられるモデルである。

特異モデルを含む一般の統計モデルに対する AIC 及び BIC の拡張として広く使える情報量規準 WAIC 及び広く使える Bayes 情報量規準 WBIC が提案されている。特異モデルにおいては最尤推測や事後確率最大化推測よりも Bayes 推定の方が自由エネルギーや汎化損失を小さくできるという意味で有効であることが数学的に証明されており，WAIC 及び WBIC は Bayes 推測のフレームワークにおける情報量規準である。一方で，これらは Bayes 推測であるがために平均という数値積分が要求され，また Bayes 事後分布に従う確率変数（パラメータ）のサンプリングも必要となる。特に WBIC については非主要項はサンプルサイズを大きくしても決して小さくはならず，数値的には分散が大きくなる。

2017 年, BIC の特異モデルを含む一般の統計モデルに適用できる形での別の拡張として特異 Bayes 情報量規準 sBIC が提案された. 日本語による記事があまり見当たらないため, 本記事ではこの新たな情報量基準の概説を行う. 本記事の構成は以下の通りである. まず第 2 章で sBIC の背景にある理論の概要を説明し, 次に第 3 章で sBIC を含む情報量規準について述べる. 最後に第 4 章で総括を行う.

◆: おことわり

本記事は全くの紹介記事 (レビュー) であり, 筆者の業績による新規性のある事柄はほぼ含まれておりません. 従いまして専門家の皆様にとっては大変つまらない記事になっているかと思われます. 申し訳ございません. また, 本記事はコミックマーケットにて頒布される予定の技術誌への投稿を前提としています. そのため, 理論を専門とされていない方にも読んでいただけますように数学的な厳密性を欠く表現易しすぎる説明がしばしば登場します. どうかご容赦くださいますよう宜しくお願い申し上げます. 一方で, 理論的な背景こそが情報量規準の強みであり, 「ラーニングや統計学, データサイエンスは不良設定問題をヒューリスティックに解く世界なので理論などなく学問と呼べるかも怪しい詐欺の世界なのだ. ぷぷい」といった誤解を受けぬためにも数学的な要素や数式はふんだんに使わさせていただきたく思います. 実際, 以下の理論的な記述はカッコツケではなく必要不可欠なものです.

第 2 章

理論

本章では情報量規準を語るに欠かせぬ統計的漸近理論及び学習理論について述べる。まず最初の節で Bayes 推定のフレームワークと性能評価に重要な確率変数を一般的に述べる。次に、Bayes 推測における統計的漸近理論を概説する。最後に、前述の漸近理論の主要項を司る係数にして代数幾何学における重要な量である「実対数閾値」について、学習理論の視点から具体的に述べる。

2.1 Bayes 推測の枠組み

N 次元実ベクトル x の確率密度関数 $q(x)$ を真の分布, $p(x|w)$ を確率モデルまたは学習モデルという。ここで d 次元実ベクトル w はパラメータでありコンパクト集合の元であるとする。パラメータの確率密度関数 $\varphi(w)$ を事前分布という。サンプルサイズ或いは学習データ数を n とするとき、サンプル或いは学習データは

$$X_i \sim_{i.i.d.} q(x) \ (i = 1, \dots, n)$$

という n 個の真の分布に従う独立確率変数である。これらを

$$X^{(n)} := (X_1, \dots, X_n)$$

と書く。

本記事では簡単のため実現可能性を統計モデルに仮定する。すなわちあるパラメータ w_0 が存在し、

$$(p_0(x) :=) p(x|w_0) = q(x)$$

が成立するとする。このパラメータ w_0 を最適なパラメータと呼び、 $p_0(x)$ を最適な確率分布（或いは確率密度関数）と呼ぶ。最適なパラメータの存在を仮定するがそれは一意とは限らない。しかし $p_0(x)$ は確率密度関数として一意である（最適なパラメータが実質的にユニークである、という）。

ガチャキング制作記 (サーバー編) と一ふとふ著

はじめまして、と一ふとふと言います。

前回の冬コミではディープラーニングの 0.5 歩目というものを書かせていただきました。

今回は、昨年 10 月から今年の 4 月にかけて、所属しているデジタル創作同好会 traP で開発していたゲーム「課金王者ガチャキング」の製作記を書きたいと思います。クライアント編として phi16 さんが同じくガチャキング製作記を書いていますのでそちらも合わせて読んでいただくと面白いかもしれません。

第1章

課金王者ガチャキングとは？



ジャンル：友情破壊系対戦ボードゲーム

コンセプト：みんなで盛り上がる楽しいゲームをしたい！

ストーリーあなた達とはあるソーシャルゲームをプレイしている人間です。そんなある日、あなたはこう思いました。-ガチャが引きたい…!!!-あなたはどうしてもガチャ欲を抑えられません。そう、ガチャを引くこと以外にガチャ欲を満たすことはできないのです…とはいえ、ガチャを引くにはゲーム内の課金要素である†秘宝石†を集めなければいけません。というわけで非宝石を集めにあなた達は暗躍するのであった…引用：課金王者ガチャキング企画書より

簡単に言うと、ガチャを引くためにバイトをしたりクエストに行ったり相手プレイヤーを出し抜いたりするゲームです。

詳しくは <https://gachaking.trap.games/manual.html> を読んで遊んでみてください！

ガチャキング制作記 (クライアント編)

phi16 著

こんにちは。phi16 です。traP というサークルで「課金王者ガチャキング」というゲームのクライアントサイドを担当していました。現在はβ版として <https://gachaking.trap.games/> で公開中です。遊んでみてください。

今回はその開発史みたいなものを書こうと思ったので、書きます。開発体制や流れとかだけではなく、設計や実際のプログラムについての話も書きます。

よろしくおねがいします。

第1章

開発の経緯



1.1 プロジェクト開始まで

traP というサークルでは、人の出した企画に対して join する形で制作が開始します。基本的には集会でのプレゼンによって入るプロジェクトを選ぶわけです。

が、ガチャキングはプレゼンとして見るより前に「自作アナログボードゲーム」として tubotu くんが見せてくれました。「課金王者ガチャキング(仮)」という名前で、ガチャを引いて満足度を得ることを目的としたゲームです。サークルのメンバーでテストプレイしたところ、その時点で完成度も高くかなり楽しいゲームになっていました。これをデジタル化してネット対戦できるようにする、という企画でした。

アナログのちょっとした悩みとして、プレイヤーの持つデータが多くなかなか管理しづらいというものがありました。チップや点棒を使いつつ、レベルをわざわざ携帯のメモに書いて記録するなどしていたのです。

このような「本質的ではない情報過多」というのは UI の改善でどうにかなる場合が多いと思います。デジタル化するとしたらこういうことを丁寧に考える必要があるというのを強く感じま

した。

私がプロジェクトに参加すればそういう点を改善できるだろう、という気持ちと、普通にゲームが面白かったので絶対に企画倒れさせたくない、という気持ちもあって。あと普通に誘われたので。

クライアントサイドのメインプログラマとして参加することにしました。

1.2 雰囲気

ゲームの雰囲気をとても大事にしたいと私は考えているのですが、その頃はまだガチャキングの雰囲気は不安定でした。ゲームプレイは殺伐としつつもわいわいという感じです。アナログで絵柄の無い状態ではあまり違和感はなかったのですが、デジタル化となれば余白が増えますし、できることも増えます。やっぱり楽しさを直接表現したいのです。

その時同時に問題点として、絵を描ける人が居ないというのがありました。プレイヤーの手札としてかなりの種類のカードがあるため、これを絵で表現できないのはちょっと厳しいものがあります。

話し合って出てきたアイデアとして、「いらすとや」を使う、というのがありました。フリーで出すものにはフリーで使えるので、素材は使い放題です。雰囲気の統一という面でも完璧ですし、絵が描けない問題も解決です。

最終的にはカードの絵柄をいらすとやにして、その外部はある程度シンプルな UI に。画面にいらすとやでできた人形劇みたいなものを表示してわいわい感を表現、という形にまとまりました。

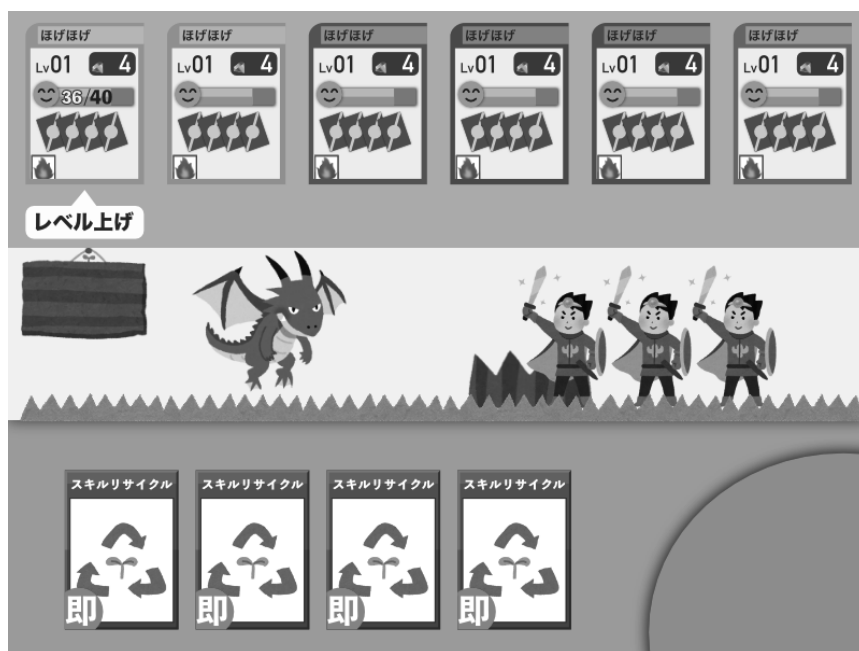


図 1.1: 画面構想

並行性への道

phi16 著

こんにちは。phi16 です。ガチャキングの開発に利用した、並行処理を司る自作ライブラリの解説です。ライブラリ自体は <https://github.com/phi16/Q> にあるので自由に見たり使ったりしてみてください。ゲーム開発については「ガチャキング制作記」を参照のこと。質問とか云々は Twitter などどうぞ。

第 1 章

継続の力

並行計算のモデルを実際に JavaScript/ES6 で組むために、まず継続の概念が必要になります。具体的には継続の構造に付加構造を与えることで実際に並行性を獲得できるようになるのです。そして継続というモデルは文法にも影響を与えます。実際、継続の力によって通常の手続き型の処理と変わらない記法を手にいれることができるのです。

全ての基本にあるのは継続です。

1.1 コールバック

JavaScript においては多くのライブラリが「コールバック」を利用していました。これは単純に、「処理の結果を関数に渡す」という仕組みのことです。

```
function process(callback){
  callback(42);
}
```

この関数は、**return** することがありません。代わりに引数に値を渡します。例えば—

```
function receiver(x){
  console.log(x*2);
}
```

を定義した上で `process(receiver)` と書くと、`callback = receiver` に引数 42 が与えられ、84 が出力されるでしょう。このように、「結果を渡される関数」のことをコールバックと呼びます。しかし実はこれは関数型界限における**継続**の概念の一種と考えられます。以後これを継続と呼びます。

1.2 継続の連鎖

A を継続に渡す関数 f 、B を継続に渡す関数 g があるとします。では A と B を使って C を返す関数 h に、その値達を渡すにはどうすれば良いでしょう。

もっと具体的な設定をしましょう。

```
function f(cb){
  cb(2);
}
function g(cb){
  cb(3);
}
```

この状況で、「 f の結果としての 2」と「 g の結果としての 3」を加算して 5 を得たいのです。

方法は単純で、 cb として g に関連する関数を渡してやれば良いです。

```
f(function(x){
  g(function(y){
    console.log(x+y);
  });
});
```

こうすれば 5 が出力されます。逆にこの記法を元とすれば継続によって処理を書く方法がわかるようになるかもしれません。「処理関数 f に渡した無名関数の引数 x 」が、「その処理関数の結果 2」なのです。

1.3 型

多くのプログラミング言語には、型があります。型のないプログラミング言語にも、仕様としての型を考えることは一応できます。

「型 A の値を受け取って型 B の値を返す関数」の型を $A \rightarrow B$ と表記します。これが最も基本で、最も複雑な概念です。

例えば `function(x){return x+1;}` は $\text{int} \rightarrow \text{int}$ の型を持つ、と行うことができるでしょう。JavaScript では数値に区別がありませんが、これは「仕様」なので型がこうである、という言明は確かに意味を持ちます。

f の型は何だと考えられるでしょうか？ まずそのためには cb の型を考える必要があります。 cb は、 int を受け取って何も返さない関数です。「値が無い」ことを $()$ で表記します。このとき、 cb は $\text{int} \rightarrow ()$ の型を持つと言えるでしょう。では f の型は？ $cb : \text{int} \rightarrow ()$ を受け取って、何も返さない。つまり $(\text{int} \rightarrow ()) \rightarrow ()$ の型を持つと言えます。

この型が非常に大事なのです。今回は int を (継続に) 返す処理だったので、 $(\text{int} \rightarrow ()) \rightarrow ()$ の型になりました。一般化すれば、A を返す処理は $(A \rightarrow ()) \rightarrow ()$ という型を持つと考えられます。

これが、継続の型です。

```
(A → ()) → ()
```

型を考えることで、自分はどのような処理を書けばいいのかが明確になります。型を考えることで考える必要のある部分を減らすのが目的です。