

このページはNYSDLで公開しています（転載可）
任意に引用したり、コピったりできます。

めぬー

[おめが?日記](#)

[おめが・企画厨隔離室](#)

[楽しくやわらかファミリー](#)

マズーはじめに

重要っぽいこと

- 多様性(量は何事にも勝る)
- 一意性(秀でた特性は他を圧倒する)
- 単純/短絡思考(シンプルさは誰にでも伝わる)
- アフォード(縛り付けるのではなく、誘導する)
- メタファー(共通知識は有効に使う)
- そして何よりもバランス

すべての情報は正しく、また正しくありません。すべての情報を信じ、疑う必要があります。この文章自体もまた正しくありません。

すべての情報には、0%より大きく100%未満の信頼性が付随します。必ず正しいことはなく、必ず間違っていることでもあります。

あなたが信頼してよいことは、あなたが今現在触れているものと、それに対する感情だけです。しかし、これもすぐさま過去となり、信頼性は失われます。

唯一重要なのは現在です。そして、次なる現在となりえる未来です。しかし、過去はただの改ざんされうる記録であり、未来はまだ存在しえぬ夢でしかありません。この文章もあなたが見ているときには既に過去のものです。

すべての文章は思考の残滓に過ぎず、思考を完全に伝えることはできません。

この文章は矛盾を含みます。

良いとか 悪いとか

何かをするときには、「なぜそれをするか」「結果何が起きるか」を考える必要があります。

「良い」とは何か考える必要はあります。

結論を言えば、「良い」「悪い」ということは、何らかの突出した特性(癖)を、ある価値観に当てはめた際の評価です。評価 = 特性 × 価値観。つまり、価値観または特性が無ければ、良い悪いといった評価は発生しません。

情報の価値は、分散値が高いほど価値が付きまゝ。(後述のパラメータ強度)
突出した特性 = 分散ですから、特性を消すのは、価値自体を潰すのに等しいといえます。没個性への道です。
対処すべきは価値観です。価値観さえ受け入れてもらえれば、どんな理不尽なルール、システムも、良い特性として理解されます。

価値観

価値観は思い込みです。

残機つぶしをして常に残機1/0で進まなければならないゲーム「バトルガレック」は、生存を第一とするシューティングとして理不尽ですが、「常にギリギリの状態を演出しながら戦うプロレスゲー」と理解すれば問題はありません。

思い込みは見た目から来ます。
同じゲームシステムでも、かわいい小動物のキャラを殺すのと、憎たらしい悪人面を殺すのでは、思い込みは異なります。

価値観は視点です。何を重要とするか、何を低く見るか。

問題解決

問題を解決するには以下の方法があります

- 問題から発生する不利益を解決する
- 問題を問題として認識しない
- 問題が発生する状況を無くす

これらの方法は時と状況で選択すべきです。

一般には一番目の手段が「問題解決」とされています。しかし、これは単純な対処であり、突進のような思考停止でもあります。ゲームルールでこれを行う場合、ルールの追加という形になるため、ルールが複雑化します。

二番目の方法は、いわゆる「一番美しいものを作った。有名な建築家が(ry)」であり「仕様です」です。こういうとアレな選択に見えますが、使う機会はちゃんとあります。あなたが問題として認識している事柄は、他の人からは問題として認識されていないことがあります。この場合、その問題を解決しても、大して効果はありません。気にするから気になるというヤツです。

三番目の方法は、問題発生原因ごと取り除くという方法です。自動車事故が起こるのは、自動車があるから、という理論です。ゲームルールの場合、ルール削除という手段になるので、ゲームは単純化されます。
問題が発生し解決方法も分からない場合、この手段は非常に有効で強力です。

解析/分解

物事を解析するには構造解析の技術があると良いです。これらの解析は普段人間が特に気にせず行っています。

プログラミング、UI設計に応用可能

- ノード
- リンク

- ツリー構造(木構造)
- リスト構造
- ウェブ構造

- カテゴリ
- 階層
- 属性

大抵のものは、単純な構成要素がいくつも重なって出来ています。複雑なことは何一つありません。複雑に見えるのは、そのシステムの要素が多すぎるだけです。この思考法は単純ですが、一方で多量の要素を扱わなければならないため、思考量が増大しすぎる傾向があります。一般に理系思考がこの形式といわれます。(ボトムアップ思考)

しかし、単純な構成要素からだけでは、予測できない効果をもたらすシステムが存在します。具体例は脳。「創発」と呼ばれる現象です。創発が発生した場合、ボトムアップ思考による解析は難しくなります。

逆に、表面から見えるシステムの振る舞いを見て、分解せずにそのまま利用することもできます。この思考法は、振る舞い全てを観察し把握するため複雑ですが、要素数について考える必要がありません。一般に文系思考がこの形式といわれます。(トップダウン思想)
トップダウンの場合、創発を意識せずに解析することができます。

過去の英知

必要と思える勉強はしましょう。過去の事例と英知は、あなたが無知であったときにするであろう無駄な時間/作業/コストを大幅に削減してくれます。車輪の再発明を防ぎ、それに費やしたであろう時間をあなたがしたいことに費やすことができます。

あなたには一人分の人力(労働量)が究極的資源として提供されています。ですが、デカイことをするにはとにかく足りません。楽をするためにありとあらゆる努力をしましょう。楽とはサボることではありません。楽とは「最大効率」のことです。

楽をするために学習法自体の研究も必要でしょう。

すべてのことを知り尽くすには、一人の人間はあまりにも非力です。効率を上げることは必要不可欠です。

可能ならば、完成しているもの、そのものを利用して良いでしょう。ただし、それを作った人がいることを忘れないでください。あなたが今いるのは、あなたへと続く過去の人がいることを忘れないでください。

そして、過去の英知を未来へ繋ぐことができるのもあなただということを忘れないでください。あなたがその英知を放置すれば、埋もれてしまい、歴史から消えてしまうかもしれません。過去を利用してください。未来へと残してください。

ゲームと作業

ゲームは作業です。特に、楽しい作業をゲームと言います。一般的に作業は面倒で、苦痛とされています。ゲームは一步間違えば、ただの苦痛マシーンです。

ゲームを作ることは、「面倒なことを他人にさせるために面倒なことをする」ことです。あなたの少しの面倒で、他の沢山の人に面倒なことをさせることができます。

あなたは、面倒なことをすることによって、他の沢山の人々の生産効率を下げ、墮落させることができます。ゲームはあなたの健康/人生/脳みそを損なう恐れがあります。

Games all bad, and make you mad.
(9bit-confusionサイトトップより引用)

あなたは、ゲームで世界を墮落させる悪の手先になることができます。さあ、ゲームを作りましょう、同土諸君。

面白さ

ゲームは無駄です。作業です。それを認めた上で、ゲームは「なぜ面白いのか」ということを常に理論的に考え、それをゲームにフィードバックして行くことが大切です。

テトリスは延々を死なないう耐えるだけの耐久レースです。なぜ面白いのですか？
シューティングは延々弾を避けつつ敵を倒す耐久レースです。なぜ面白いのですか？
RPGとSLGは延々数字いじりをするゲームです。なぜ面白いのですか？

シンプルと複雑さ

シンプルは入り込むときの敷居の低さです。
複雑さは入り込んだ後の奥深さです。
これらはかなり近い位置にいますが、同一のパラメータではありません。つまり、シンプルでないのに複雑でもないゲーム、シンプルでも奥深いゲームは存在します。

ゲームデザイン

ゲームデザインは以下の要素からなります。

- ゲームシステム
- ユーザーインターフェイス(UI)

ゲームシステムは数字とロジックのやり取りです。入力を受け取り、内部の数字/式と演算し、出力を返す。ただそれだけの処理系です。人間の意味感情とは関係性はありません。

数字とロジックだけの無機質な存在です。

ユーザーインターフェイス(UI)はゲームシステムとプレイヤーの橋渡しです。ゲームパッド、マウス、センサなどの入力系と、モニタ、サウンド、フォースフィードバックといった出力系、その他、マニュアル他から成ります。これらは無機質なゲーム世界のパラメータ/トークンに、人間世界における意味づけをし、プレイヤーへの情報伝達を行います。

数字とロジックの世界

システム、系から見る

必須事項はこれだけ

- フィールド(2Dマップ/3Dマップ他)
- トークン(キャラクタ/自機/ユニット)
- ルール

系のうち、ルールに「終了条件」と「勝利条件」が含まれる場合、いわゆる古典ゲームになるようです。終了条件はゲームオーバーであり、勝利条件はクリアです。つまり、初期状態でフィールド/トークン/ルールが満たされる、現実世界は、たった二つの要素を足すだけで、ゲームになりえます。

ルール

ルールは以下の要素からなる対です

- 一つ以上の「発動条件」
- 一つ以上の「効果」

パラメータ

パラメータは数値です。それ自体は一軸の「値」を持ちます。パラメータは演算が可能です。複数のパラメータをカプセル化することで、より高次のパラメータにすることができます。(例・ベクトル/行列)

パラメータの強度と多様性

パラメータは一軸の「値」です。表現方法が文字でない(コンピュータゲーム/ボードゲーム)場合、値は離散です。つまり、nパターンの値しか保持できません。

ゲーム中、パラメータにはなんらかの意味がつきます。意味の強度は、多様性に反比例します。つまり、値が0/1しか存在しない、ブール値は意味が最大になります。(例・成功/失敗)ですが、ゲームに必要な多様性は2パターンしかないことになります。逆に、十分に大きい範囲で値が取れる場合、意味はゆるくなります。

ゲーム判定

ゲームにおける最終的な判定は、スレッシュホールド(閾値)しかありえません。つまり、なんらかのパラメータと、その閾値、その判定結果 = 成功/失敗です。
このパラメータは単一のパラメータではなく、複数のパラメータやパラメータの合成(関数変換/写像)でもかまいません。

判定とパラメータの多様性

「判定において、スレッシュホールドの幅を大きく取る」ということは、前述の「パラメータの多様性をどれだけ許すか」ということです。
多様性を許さないとタイトな、ひたすら平均台の上を走るようなゲームになります。逆に多様性を許すとゆるい、ぬるま湯のようなゲームになります。

ゲームの深さ

ゲームはプレイヤーに多様な状況を提供します。それに対して、プレイヤーはゲームに多様な入力を返します。
プレイヤーに提供する状況がそれぞれがユニークで、尚且つ、プレイヤーが返すべき最適入力(最適解)が状況ごとにユニークであれば、深みがあるゲームとなります。所謂STG的なやりこみです。スコアを上げるために、ギリギリのカスリ点までも狙う。入力最適化までの道のりが長い。それが深みです。

最適入力は状況毎に存在します。そうすると、個々の最適入力までの深み×状況数がゲーム全体の深さになります。

個々の最適入力までの深みは、ゲーム内パラメータを複雑化させればよいだけです。状況数は、単に膨大な量のデータを詰め込むだけです。
プレイ時間は有限ですが、比較的簡単に大きくすることができます。

スキルと戦略性

スキルは短期的な技能です。大抵の場合、スキルに成功するとリターンがあります。
リスクとリターンは噛合う必要はありません。ローリスク&ハイリターンなら「必須技術」になり(弾を見たら避けるとか)、ハイリスク&ローリターンなら「ネタ技」、ハイリスク&ハイリターンなら「一発逆転/上級者向け」(ガレツガ臨死/ブロッキング)になります。

技術力、慣れなどで、リスクが極端に減っていき、ハイリスク&ハイリターン　ローリスク&ハイリターンになるものを持つゲームは、テクニカルになります。例・近接ショット、ケツイ5箱
この手のものは、リスクとリターンが噛合っていないために発生する仕様です。

一方逆に、完全にリスクとリターンが噛合っているものもあります。長期的なスパンで起きる戦略性です。戦略ゲーにおける資源分配、シューティングにおけるボム分配などの、いわゆるリソースコントロール(資源管理)です。
これらは、可能な限りの組み合わせから、最適解を探すことです。ですから、スキルの様に、慣れていないため失敗する　というようなことは発生しません。ただし、資源管理中に含まれる行動がスキルを伴う場合は、この限りではありません。

プレイヤーから見る

対話

プレイヤーとゲームシステムの対話、これは人と人の対話と比較できます。人と人の対話には、ターンが存在します。同時に二人が言い合う状況は会話では存在しません。それはケンカです。

対話をする場合、二つのモードを使い分ける必要があります

- 話すターン(ゲームシステムから見た入力、プレイヤーから見た出力)
- 聞くターン(ゲームシステムから見た出力、プレイヤーから見た入力)

このターンのサイクルが対話です。サイクル速度は任意ですが、両者が適応できる速度であるべきです。

ターン性ゲームの場合、ターン終了時間をプレイヤーが決定できるので、ターンサイクル時間は任意といえます。アクションゲームの場合、ターンサイクル時間は固定で、16.66msec(60fpsの場合)といった極端に短い時間ですが、時間が短いため大抵のプレイヤーはターンサイクルであることを意識しません。代わりに、反応が早いという特性から「リアルタイム」という概念を導入して認識していません。

スーパープレイの予測、学習過程

ゲームは学習です。一人用ゲームの場合、プレイヤーの学習過程を予測しておくべきです。

少なくとも押さえて置きたいのは、「初回プレイ」と「熟練しまくったスーパープレイ」のイメージです。

「初回プレイ」は導入です。稼ぎプレイなどの熟練プレイに対して簡単(レベルデザイン)で、フィーリングでそれなりにプレイできる(事前知識/特殊な技能を必要としない)と良いでしょう。

「熟練プレイ」は最終的な目的地です。できればカッコよく、直感的に(一般的な価値観で)「上手いプレイ」と思えるスタイルが良いです。が、別に価値観は演出やスコアリングで変更できるので、一般的価値観に引きずられすぎないように。

この二つのプレイが確定し、その間がある程度緩やかに連続的であれば(ある程度の学習/熟練でプレイを上達可能なら)、ゲームプレイはキレイなゲームになるでしょう。

連続的というのは、先がある程度見える、ということです。どうすれば上手くなるかが分からないと、プレイヤーは学習できません。とりあえずの方針を何らかの方法でプレイ中に示すと良いでしょう。

ただし、すべて先が見えてしまうのも、人間にとっては退屈です。たまにはサプライズが必要。

アフォード/メタファー/UI

事前知識

人間は多くの前提知識を、事前学習によって脳に蓄えています。また、やたらと高度なルールを前提知識として持っています。ゲームをより簡単に理解させるためには、これらを使うべきです。

事前知識とは、常識です。ただし、一般用語の言うところの常識ではありません。

物理法則と事前知識

ボールを壁に投げつけるとどうなりますか？ 当然、ボールは壁でバウンドします。これはルールです。「トークン『ボール』とトークン『壁』が接触する」ことを発動条件とし、「トークン『ボール』の移動ベクトル成分のうち、壁の法線ベクトル成分を反転させる」ことを効果とする「ルール」です。

どうですか、このルールは文章化すると複雑に感じるはずですが、しかし、普通に説明する場合はどうですか？ 「壁に当たるとバウンドするよ」これだけで、上の複雑なルールが伝わります。

物理法則は非常に難解なゲームシステムを持っていること、その物理法則は大抵の人が常識として知っているという強力な特性を持ちます。

グラフィックと事前知識

人間キャラクタの前に「食べ物」の絵があったらどう考えますか？ 食べ物は実世界で見慣れているので、食べ物に関連した動作を容易に連想することができます。

適切な絵をゲームトークンに貼り付けることにより、そのトークンの役割をより分かりやすくプレイヤーに伝えることができます。

ゲーオタと事前知識

ゲーオタは沢山のゲームを知ってます。それを利用して、知っているであろうゲームに含まれる名称を利用することで、ルールを分かりやすく伝えることができます。(例・このシステムはhogehogetteゲームのfoobarってシステムと同じだよ。)

ただし、当然ながら、マイナーゲームであったり、相手がゲーオタでないと使えません。

マジックナンバーとカテゴリ、ボタン

人間がすぐさま覚えられる要素は、一般的に7-2と言われる。これをマジックナンバーと言う。(←は個人差)

つまり、このマジックナンバーにゲームの要素(ゲームルール、インターフェイス)が収まるのが望ましい。しかし、要素を削ればそれだけ多様性が奪われ、短期的なプレイ時間の短いゲームになってしまう。

これを解決するのが解析/分解で示したカテゴリ/属性である。
ストIIをベースとする格闘ゲームは一般的な他のゲームに比べ使用するボタンが多い。それでもなんとかなるのは、全て同一のカテゴリに割り振られているからである。つまり、全てのボタンが攻撃だ。

(歴史的にはストIの圧力センサ×2を通常コンパネにしたため6ボタンになったと思われる。) 同様に全てのボタンが攻撃である鉄拳、ギルティギア(イスカ除く)も4、5とボタン数が多めだ。

一方、バーチャファイター系は、ボタンに「ガード」という攻撃でないカテゴリの操作を加えてしまった。それ故、ボタン数を3に絞らなければ無理であった。

アーケードゲームではコンパネは水平で、全てのボタンが同一に扱われるような配置になっている。一方家庭用ゲーム、特にSFC以降はL/RボタンやZボタンという特殊な配置を持つボタンが登場した。

これはボタンの配置自体が、ボタンをカテゴリ分けしている。このカテゴリ分けに会うよう、ボタン配置することで理解しやすいゲームになる。

関連

[ねここ](#)

究極的なUI

UIはゲーム評価において減点効果の比率が圧倒的に大きい。これは、UIがクソでけなされるゲームの数と、UIが優れて評価されるゲームの数からも見て分かります。

究極的なUIは、現実の空気や水や安全のように、誰からも気にされないものだと思います。独創的な部分が極端になく、余りにも単純で、快適だから誰も気付かない。そして、どこにでもある

データ主体

誰のデータを軸に持ってくるか

製作側データ主体

製作側が用意したデータを、プレイヤーが楽しむという古典的なスタイル。物語、音楽、映像、芸術といった「受身」のエンターテイメントから、ゲームが受けついで手法。(ただし、芸術が完全に受身というわけではない。)

全てのデータを製作者が作らねばならず、そのデータ量と分散が製作者に束縛されるが、それゆえに質の高いデータを提供することができる。

プレイヤー側データ主体

TRPG的な、Web2.0的なスタイル。コントラクションゲー。

前述の受身の芸術の場合、作品にプレイヤーが介入するには、プレイヤー自身が製作者になる必要があった。しかし、入力を任意に保存/フィードバック可能なゲームの場合、プレイヤー自身が製作者にならずとも、ゲームに介入できるようになった。

データ製作をプレイヤーに任せられることができるため、製作者がデータにかかるコストを減らすことができるが、データの質の保証がなく、また、制御不能な結果をもたらすことが多い。また、製作主体側と比べて、プレイヤーに要求する入力量が増える(プレイ時間が長い/複雑)ため、マニアックな仕様になりがち。

プレイヤー同士のデータを混ぜ合わせる

新の2.0スタイル。プレイヤーが製作した(または気が付いたら製作されていた)データを他、のプレイヤーに利用させる。

相互的に補完し合い、効果が相乗的に増加するのが特徴。爆発的に増えるため、しばしば管理不能になったり、著作権上で問題が発生することがある。

製作者は基本UIとデータベースを作るだけで済む。

開発

言語

使ったことのあるやつ

- Logo, N88BASIC

へんじがない ただの こてん のようだ

- VisualBasic

ウインドウズUI, いべんとどりぶん

- VisualC++ + DirectX

はやい, つおい, ポインタのじごく, ぬるぼ

- HSP

おそい, らくちん, グローバル変数のあくむ

- JavaApplet

ブラウザできらく, OOLでキレイ, GC, しかしぬるぼ

- D言語

はやい, Cにべったり, OOL, GC付, ネイティブ

プログラミング言語に慣れていない人はHSPがいいんじゃない？インスコも言語仕様も楽に使えるように小さく手軽。逆を言うと、難しいけど使えるものも削られてるので、その辺はできない。

JavaはC++の影響を強く受けたOOL(オブジェクト指向言語)です。ガベージコレクタがあるので、メモリ管理のような面倒なことをする必要がありません。ただし、内部ではバリバリのポインタ仕様なので、ポインタに関する知識が必要です。OOLとしては洗練されているので、OOを学ぶにはJavaが良いと思います。コンパイラ/ドキュメントがフルパッケージで配布されているのも魅力です。

C/C++は現行最速最強最凶の低レベル言語です。言語としての汎用性と下位互換、最新技術への対応、政治的ないざこざとありとあらゆるモノにさらされ続けているため、既に何がなんだか分からない状態になっています。MSとか独自拡張してるし。

拡張もいたるところで、各自ばらばらにやってるので、何を選択するか幅がありすぎ。開発環境/コンパイラも同様。もう何がなんだか。

言語としては、C言語の拡張/上位互換としてC++という形になっています。まとめてC/C++。動的メモリ確保、ポインタ、OOL、アセンブラの組み込みと、やりたいことはすべてできますが、逆に言うとなんか何をするにもすべて自力でやる必要があります。それが、ライブラリの利用。高級言語なのに、バイト/ビット単位の演算とかもう(ry。これが低レベルといわれる所以。それでも、速度的にも対応範囲にしても現最強の言語であることは事実。

D言語はなんだか新しい言語です。C++/Javaなど近年の言語からいいとこ取りをした言語なので、ほぼC++/Javaのように書くことが出来ます。C言語に匹敵する速度を持ちながら、Javaのようなガベージコレクタがあるため、メモリ管理で悩まされることはないでしょう。テンプレートや連想配列といったことも可能。独特なのは、高級言語並みの文字列処理が可能なことでしょうか。現状D言語が利用できる環境はあまり多くなく、各種ライブラリも他の汎用ライブラリをポーティングして利用している状態です。C言語との互換性が高く設計されているため、ライブラリの利用自体は楽ですが、C言語の知識が必要でしょう。とにかく新出言語なので、いろいろと情報が不安定だったりするのが一番致命的なところだと思います。

プログラミング

プログラミングは、文章を書くことに似ています。言語だし。同様に「良いプログラムを書くこと」は、「良い文章を書くこと」に当てはまります。プログラムが書けることは、良いプログラムが書けることとは同一ではありません。

オブジェクト指向(OO)

ソフトウェアを部品化して、レゴのように組み立てようとする考え方。共通部分を、スーパークラスとして定義するため、ソースサイズも小さくなります。適切に設計すれば、それぞれの部品の内部が隠蔽されるため、それぞれの部品内部を考えずに利用したり使いまわしたりできるようになるので、効率が上がるといわれています。

多用される設計法は、抽象化されデザインパターンと呼ばれています。この辺はプログラミングに慣れてきたら勉強しておくとか吉。ただし、抽象化されたデザインをちゃんと応用できないなら、後回しにするべき。まずは動くこと最優先。

精神

エンターテイナーとして

ゲーム製作者はエンターテイナーです。エンターテイナーとは芸人です。我々は芸人たるべきです。人々を笑わせるために、悲しませるために、怒らせるために、でっち上げの情報を流す。そういう道化師であるべきです。

いたずら心とゆるい思考を大切に。

生産と管理

より効率よく生産するための管理技術。ソフトウェア工学。
まあ、正直一人でやる場合はあまり気にせず、参考程度に抑えた方がいいでしょう。求める規模にもよりますが、個人製作モノというのは得てして小規模のハズです。ここで述べる生産方法は基本、大規模であることを前提としています。

ウォーターフォールモデル

抽象的な生産工程を切り分けてラベリングし管理しやすくするのが、モデル化を導入する目的。一番単純かつ高速な工程。工程の後戻りはない。

設計・開発・テスト・出荷

スパイラルモデル

ウォーターフォールを数サイクル繰り返し、それぞれの各工程でリソースの引継ぎをしない。当然、生産コストはそのままn倍になるが、任意のサイクル終了時にプロジェクトを破棄できるため、リスク分散ができる。基本、生産物の精度を高める手法であり、コストは高い。近年のゲーム開発におけるプリプロ(Preproduction/製品開発の前に重要なシステム部分を作ってテストすること)も、このモデルの一つである。

設計・開発・テスト (1サイクル目)
設計・開発・テスト (2サイクル目)
設計・開発・テスト (3サイクル目)
任意タイミングで出荷

- Logoが追加。この頃からゲームしか作ってなかった -- るーと雨 (2006-01-17 00:38:46)
- なんか不思議な箇所に投稿されてしまいましたよ -- るーと雨 (2006-01-17 00:40:02)
- 強引に修正してみた -- (° ▽ °)ノ (2006-01-18 22:01:41)

名前:

コメント:

投稿

[すべてのコメントを見る](#)