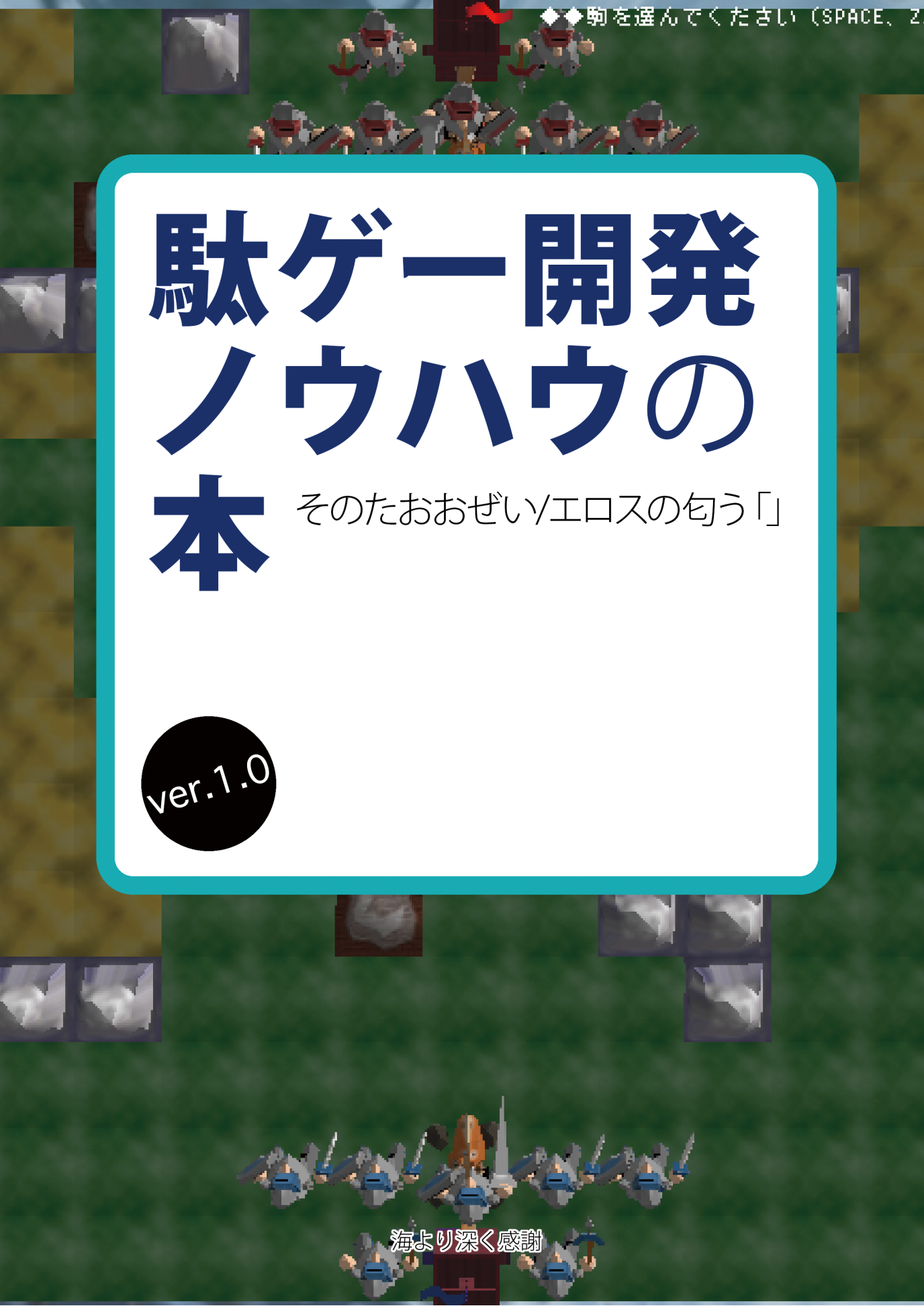


駄ゲー開発ノウハウの本

そのたおおぜいソエロスの匂う「」

ver.1.0

駄ゲー開発ノウハウの本

そのたおおぜいソエロスの匂う「」

ver.1.0

海より深く感謝

まえがき

自作ゲームを作りたい。

これは、数多くの「」が何となく、ぼんやりと持つ願望の一つであり、また、多くの「」が様々な形で手を付けつつ、なかなか到達することできない狭き門でもあります。これは自作ゲームスレで、スレが伸びる割に、完成品したアプリを目にすることはさほど多くはない事からも、よくわかるでしょう。

その一方で、自作ゲームスレは、時期によってその伸びは異なっても、絶えることなく立ち続けるスレの一つでもあります。

ゲームを作るということは、絵を描く事と同様に自己表現の一つであり、それぞれ理由や目的は異なるでしょうが、そのためにゲームという手段を選んでいるわけです。

では、自作絵に比べて、自作ゲームの完成品を見ることが難しいのは、何故なのでしょう？

それには、まずゲームを開発するという事は、自作絵に比べて完成させるまでの手間が大きく、また自作絵と違って、ノウハウの共通化が困難という点が大きく関わってきています。

例えば、「絵の描き方」でネット検索をかければ、様々な絵の様々な描き方を知ることができます。その中から自分にあったものを選ぶこともできます。目にした技法を試すのも、それほど難しくはありません。ところがゲーム開発においては、目にしたノウハウを利用するためには、開発環境や言語、開発機材を揃えて……という手間が必要となり、ある開発環境で役立つことが他ではまるで役に立たない、といった事も多く見られます。

しかし、本当にノウハウの共通化は難しいのでしょうか……？

本書は、自作ゲームスレを数多く閲覧する中で、ゲームアプリ開発の中で遭遇しやすく、また、創作意欲を萎えさせやすい幾つかのポイントをピックアップし、それらに対する可能な範囲での解決策を提示していく事を目的に書かれています。

一方、ゲーム開発環境は対象とするハードや、そこで利用される言語は多岐にわたっており、それぞれに条件が異なります。そのため、一つの環境、一つの言語のみをターゲットにすることは、ゲームアプリ開発を目指す数多くの「」のうち、ごく一部にしか利用できないものとなってしまいます。

そこで本書は、いくつかの擬似コード以外のソースコードは一切掲載せず、ただひたすらに「ものの見かた」と「考え方」を説明することに重点を置いています。

そのため、自作ゲームアプリを構築していく中で必要となってくる構成（フォーミュレーション）を、限られた範囲で詳しく説明する一方、本書に従うことで即解決（ソリューション）に直結する部分は、残念ながらほとんどなくなってしまっています。

自作ゲームのための構成（フォーミュレーション）…「ものの見方（捉え方）」「考え方」に関する解説は、一般的な開発関連書籍では短い一章にまとめられるか、あるいは全く触れられない事がほとんどです。

また読者の側も API の利用方法や、「ある問題に対するアルゴリズム（ソースコード）」そのも

のといった、「自分のプログラムにそのまますぐに引用できる情報」ではないため、読み飛ばしてしまう事が珍しくありません。

ところが「即利用できる情報」は、自分のアプリの仕様に沿ったものである事は、まずありません。こうした部分は、自分のプログラムに併せて改造する必要があります。さらに、誰かから渡された、あるいは自分で書いた仕様書に、手直し無しですぐ使えるソースリストが添付されているなんて事はありえない、というのはすぐ分かるでしょう。

また快調に作業が進んでいるときはともかく、一旦何らかの原因で引っかかってしまうと、「即引用可能な情報」…ソリューションが解決の手助けになる事はまずありません。ソリューションは、一発で全てを解決してくれる銀の弾丸、迷宮をさまよう中で道標となるアリアドネの糸のように見えて、その実、問題の原因を覆い隠し、解決を先送りするその場しのぎでしかなかった、という場合が多いのです。

こうしたソリューションを真に銀の弾丸たらしめるのは、自分自身がそれまでにアプリを構築していった経験です。

開発環境という銃に、情報という火薬を詰めて、弾丸を込め、目標に狙いを定めて引き金を引き、初めて真価を発揮できるのです。

それゆえに、本書を読んだ上で、ゲームアプリを構築していくために最も必要となるのは実践です。本書は筆者が携わってきた（それほど多くない）開発や、無数の失敗、そしてスレの閲覧の中で得た知識や経験を開陳したエキスのようなものではありますが、本書を読めば自動的にゲームが完成するわけではありません。実際に手を動かし、頭を捻ってコードを書き進めていかなければならないのです。

本書や他の参考書籍という理論と、「」の行う実践が両輪となって初めてゲームアプリが完成していくということを、しっかりと頭においておいてください。

そして、つらい、時に笑える失敗を何度も繰り返していく中で、本書の考え方を越えた、自分なりの何かが必ずつかめるはずです。

■本書を利用するにあたって必要なもの

本書はその性質から、単なる読み物としても読むことが出来ます。その場合には、本書だけで完結することになります。実際の開発に利用したい場合は、次のようなものが必要になります。

- **ターゲットマシン(※1)とホストマシン(※2)**
- **Windows Visual Studio や Xcode、Wolf Editor、ツクール系、スクリプト言語など任意の開発環境**
- **開発環境及び開発言語の関連書籍、もしくは開発者向けサイトの情報**

※ 1 作成したゲームを動かすためのシステム、Windows マシンや Android 端末など。

※ 2 開発環境を動かすためのシステム、ターゲットと一致している場合もあります。

CONTENTS

まえがき…… 3

1・ゲームとは何か…… 7
ゲームとは何か？・7 ／全ては状態が支配する・7 ／じゃあ実際の処理はどうなるの？・9 ／画面上に必要な情報・11 ／「状態の入れ子構造」・13

2・思いつきを形にするために…… 14
集めるだけでは形にならぬ・14 ／どこまで想像してますか？・14 ／実際の所どうするの？・16

3・アイデアのまとめ方…… 20
2 × 2 で整理する・20 ／ボトムアップとトップダウン・20 ／分類と整列・21 ／付箋で整理・22

4・紙の重要性…… 23
紙に書け、プリントアウトせよ･23 ／光り物は足が早い･23 ／デバグの友は紙とペン･25 ／ペーパーコーディング…仕様書の重要性・26 ／相互補完を意識する・26

5・絵も描けなけりゃ音もない…… 28
無い無い尽くし、無いつくし･28 ／マウスが友達！・28 ／駄コラ感覚で行け！・28 ／フリー SOZAI を使う・29 ／フリーのアプリを使ってみる・29

6・「簡単な方法」という王道 …… 31
枯れた方法を使い倒せ・31 ／絞り込むことの重要性・32

7・オリジナリティとは…… 34
オリジナリティが分からない・34/ オリジナリティって何だ？・34 ／○○の焼き直し？・35 ／借り物しか作れない、借り物しか見てくれない…・35 ／ゲームとレポート・37 ／要素の捉え方・39 ／要素を再構築するときの注意・40 ／ばっさり切ってしまう事も必要・42 ／「今週の目玉」に注力する・42

8・始まりの終わり…… 44
守破離の「破」にはまだ遠い・44 ／それがなぜ完成しないのか？・44 ／アプリは経験で作るもの・46 ／気負わずに行こう・46 ／心を映す工芸品・47

補遺・最初の一歩のその先に…… 49
作れるようにはなったけど……・49 ／サンプルコードを読み解くには・49 ／磨くべきは技巧ではなく発想・51 ／論理学に学ぶ要素の切り出し方・53

あとがき…… 56

part1 ゲームとは何か

■ゲームとは何か？

さて、頭に浮かんだアイデアやイメージ、他の「」には譲れない妄想や、とりとめもない空想をゲームアプリの形に落とし込むためには、様々な情報が必要となります。

例えば、開発ツールそのものの使い方や、開発に用いる言語で使われる API、ゲーム中の様々な挙動を表現するための各種数式などなど、列挙に暇がありません。開発入門書から専門書に至るまで、ゲーム開発に関連する書籍やサイトではこうした部分の説明・解説に内容のほとんどを費やしています。

その一方で、ほとんどの書籍では全く触れられていない部分があります。

それは「ゲームとは何か？」という点です。

「そんなの説明するまでもないじゃないか！ゲームはゲームだろ！？」と思われる方も多いでしょうし、「代償行為」とか「ストレスの解消のために使われる云々」といった心理学的、あるいはゲーム理論などの経済学的な専門用語が頭に浮かぶ方も少なくないでしょう。

確かにそれらは遊ぶ側…プレイヤーから見れば、あるいは学問的な見地からすれば間違いではありません。しかし、ゲームアプリを構築していく側から…「プログラムの側から見たゲーム」とは、一体何なのでしょう？。

結論から言ってしまえばゲームは内部状態の集合体…「ゲームとは状態である」ということが出来ます。

「ゲームは状態」と言われてすぐ脳裏に「！」が浮かぶ方は本書を閉じ、すぐに開発環境を立ち上げてアプリの構築に取り掛かれるはずですし、今すぐそうすべきです。ですが、ほとんどの方は、なんじゃそりゃ？と思われるか、言われてみれば何となくぼんやり……といった感じでモヤモヤしたものが脳裏をよぎっているはずです。

ではなぜゲームが状態なのか、詳しく説明していくことにしましょう。

■全ては状態が支配する

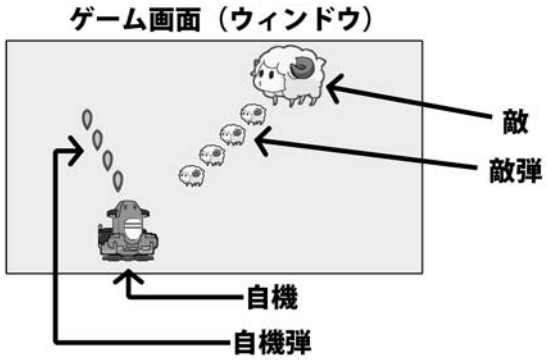
先に述べたとおり「ゲームは状態である」と言われても、なんだかよくわからないのは当たり前。そこで理解しやすく説明するため、一つの例として架空のシューティング・ゲームを考えてみることにしましょう。

シューティング・ゲームは、横スクロール縦スクロール、弾幕、ＦＰＳと様々な種類があります。しかし細かい演出その他を考えず、要素だけを取り出して単純化すれば、画面上に自機と自機の弾、敵と敵の弾が表示されているのはほぼ共通しています。また、そのルールを考えると自機の弾が敵に当たれば爆発、敵の弾が自機に当たれば爆発。という極めてシンプルな構造になっています（fig.1-1）。

では、画面に表示されるものの中から自機を例に取って考えてみましょう。

自機は画面上に表示される、プレイヤーが操作可能なキャラです。自機はプレイヤーが操作し

Fig.1-1　シューティングゲームの模式図



た結果を反映させるために、画面上に表示される必要があります。そのためには、最低限画面上に表示するための座標を必要とします。また、プレイヤーからキーボドやジョイスティック、画面スワイプなどの何らかの操作を受け取って移動しますが、それがどのような操作であるのかの情報を保存しておく必要があります。敵弾が当たれば操作不能になって爆発させなければなりませんから、そのための情報も必要になります。

これら「自機の操作や表示のために必要な情報」は「自機の状態」と言い換える事ができます。

単純な移動と弾の発射以外にも、ゲーム開始直後に登場したり、アイテムをとってパワーアップしたり、無敵状態になったり、敵弾や敵機、地形にあたって爆発したりと、様々な状態を考えることができます。

もちろん自機の弾や、敵、敵の弾にも同様の情報…「状態」があり、画面上のキャラは全てこうした状態を持っていることになります（fig.1-2）。

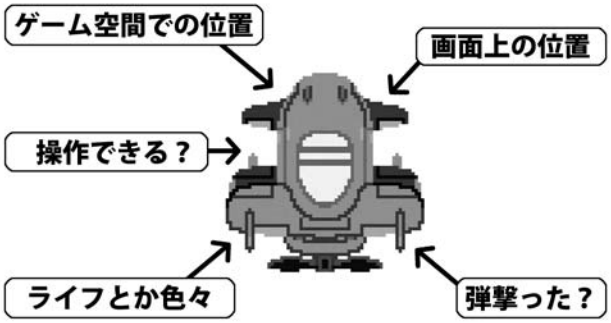
さらにこうした画面に登場するのすべてのキャラは、「ゲームの状態」の元に動いています。「ゲームの状態」とは、ゲームが今遊ばれているのか、デモ中なのかなどの情報を含んでいます。

例えば「ゲームの状態」がデモ中であれば、自機は表示されませんし、デモプレイであれば画面上で自機は動いていてもプレイヤーの操作は受け付けません。

プレイヤーからの操作も「状態」の一つです。自機を動かす場合、プレイヤーからの操作という「状態」やゲームの「状態」を元に、自機の「状態」を更新していくのです。

このようにゲームの内部には様々な状態があり、その積み重ねが「ゲーム」の骨格をなしていると言っても良いでしょう。

Fig.1-2　自機（キャラクタ）が持つ「状態」



ではシューティングではなくRPGやシミュレーションのようなゲームの場合はどう考えていけばよいのでしょう。

シューティング・ゲームの場合は「画面上に登場する（描画される）キャラクタにそれぞれ状態を持たせる」という、直感的に分かりやすい形に落とし込むことが出来ました。ところが、コマ（ユニット）単位で分けられそうなシミュレーションはともかく、フィールド型RPGの様な、フィールドと戦闘画面が切り替わるようなゲームでは、そうした方法が通用するようにか思えません。ではどのように考えればよいのでしょう。

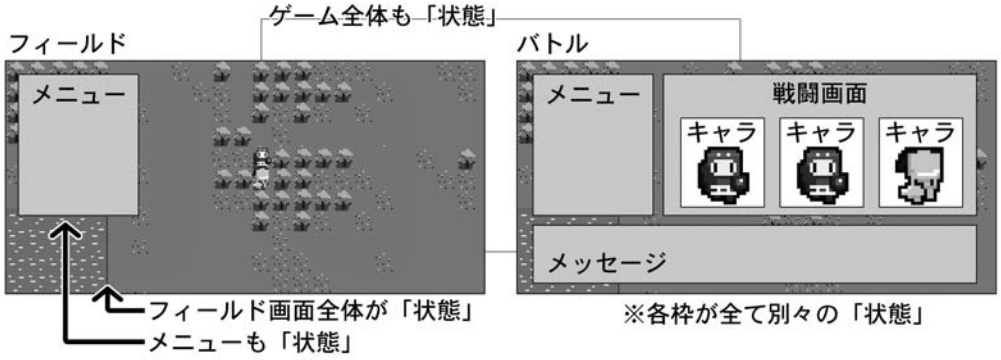
これはもちろんプログラムする人の考え方によって大きく変わりますが、「ゲームの状態」の中に「フィールド」と「戦闘」という２つの別々の状態を作り、さらにその中に「画面上のどの位置に何を表示するか」という状態を作ると分かりやすくなってくるでしょう。

例えば「戦闘時のメニュー」や「モンスターが表示される部分」「ゲームからのメッセージ」が表示される部分をそれぞれ状態として持たせるようにするのです（Fig.1-3）

マップや、キャラクタごとの状態は、「ゲームの状態」の中に収められ、必要に応じてその状態を更新していくという事になります。

ここまで「ゲームの本質は状態の入れ子構造」である（そう考えると分かりやすい）、という事を説明してきましたが、では、自分のアイデアをこうした「状態」に分解していくにはどのような考え方をすればよいのでしょうか？

Fig.1-3　RPGも「状態」の積み重ね



■じゃあ実際の処理はどうなるの？

「ゲームの本質は状態の入れ子構造」であり、ゲームを構築するためには、自分のアイデアを「状態」に分解して、プログラムに落とし込んで、アプリを構築していけばいい……と、言葉にすれば簡単ですが、それだけだと何がなんだかわかりません。

実際には、どのような作業をすればよいのでしょう？、ザックリと紙に書き出したアイデアをどのように「状態」に分解し、それをプログラムにしていけばよいのでしょうか？

ここで必要になってくるのは、さらなる分解です。

先程書いたfig.1-1を見てみることにしましょう。これは先の述べたとおりシューティングゲームの簡略化されたイメージです……が、なぜシューティング・ゲーム（STG）なのでしょう？。

それには STG の持つ特徴が関わっています[※1]。以下に列挙してみましょう。

- **他の種類（アクションゲームや RPG、ビジュアルノベルなど）のゲームにくらべて、プログラム以外に必要となるデータ（キャラ画像や効果音など）が少なく、比較的小規模なものでも完成したゲームにすることができます。**
- **未完成の状態でも動作させやすく、結果がすぐ目で確認できることができます。「コツコツ派」も「いきなり派」も結果がすぐに得られるので、プログラム意欲を持続させることができます。**
- **リアルタイムゲームの特徴からユーザーからの反応を可能な限り素早く受けとり、反応を返さなければなりません。そのため、アプリ全体の高速化に気を配る必要があり、アプリ全体のボトルネックを調べたり、無駄な処理を減らす練習になります。**
- **「描画」「入力」を必須とし、「音」、さらに必要に応じて「ファイル操作」が加わるため、プログラムの基礎がすべて詰まっています。**

駄ゲーの道を進もうとする時、目指す目標が RPG であれ、シミュレーションであれ、ノベルゲーであれ、パズルであれ、まず STG を構築して、「ゲームアプリケーションの全体の構造」を把握しておけば、その後の手間を大きく減らしてくれます。

さて、アイデアとして頭に浮かんだ大雑把な単純化されたイメージを「状態」に落とし込むためには、それらを詳しく細かく分け、要素を考えて、紙に書き出していかなければなりません。コードに落とし込める程に細かい「状態」に切り分けていくのです。

これは、目標をはっきりさせるためにも、途中で諦めないためにも、細かい部分での見落としを避けるためにも必要です。

さて、fig.1-1 を基にして、ゲームのルールやアプリケーションに必要な機能を考えてみました。

○アプリケーションのアイコンをダブルクリックすると起動する
○もちろんデモなどない、タイトルも面倒くさいので出さない
○いきなり始まる
○モデルや画像のファイルを読み込んで表示し、これをキーボードやパッドなどで操作できる
○プレイヤーが操作できる自機があり弾を発射することができる
○敵が存在し、これは自機の弾が当たると爆発して消える
○敵も弾を発射し、自機が敵の頭に当たるとやはり爆発して消える
○弾は画面外に出ると消える
○自機も敵機もやられると一定時間後に復活する
○復活位置はランダム
○背景がある
○効果音は未来への遺産

※ 1 シューティングゲームは構造が比較的単純で、ゲームを完成させやすい。という利点がある一方で「他のゲームの差別化や、演出に重点を置くあまり、プレイヤーに対して特に意味も無く高度な技巧（弾避けの技術など）を求めてしまう」という欠点もあります。こうした方が、プログラマや演出をする人にとって楽なのも事実ですが、こうした部分こそ手を抜かずしっかりと作っていくべきでしょう。

○ BGM が漢字で書けない
○ゲームオーバーは無い
○ウィンドウのクローズボックスをクリックしたり、ウィンドウを閉じるキーボードショートカットを押すと即終了する

誰かに説明するときには、ほとんど「出た見た撃った」で終わるシューティング・ゲームも、実際にルールや必要な処理を言葉で説明するとこれほどの量になってしまいます。

では、こうした目標をさらに細かく分類していって、「状態」との接点を見つけ出すところまで分解していくことにしましょう(※1)。面倒臭い作業ですが、慣れれば意識せずに行えるようになりますのでご安心を。また、これをきちんと行っておくことで、途中で迷ったりする事がなくなり、後の作業がぐっと楽になります。

■画面上に必要な情報

スケッチを元にして、まず気が付くのは「画面上に何か表示させる必要がある」という事です。そして fig.1-1 や先に挙げた細かい目標を調べていくと、画面上に必ず表示させなければならないものは次のようになります。

○自機
○自機の弾
○敵機
○敵機の弾

とりあえずは、この四つを表示することができれば、ゲームとしては成立します。
またこれらは全て「状態」で表現することが出来ます。具体的に挙げてみましょう。

○画面上に表示する時の座標
○画面上に表示される画像（読み込むためのファイル名など）
○どのような状態にあるか（爆発か、操作可能かなど）

最低限これだけあれば、自機や敵の弾といったキャラクタを、画面上で動かしていくこととが可能です。

そして、この四つがきちんと表示できるようになり、曲がりなりにも操作できるようになった後に、次のものを表示させる事を目指すことになるでしょう。

○背景
○文字情報（得点など）
○爆発したときの効果

※ 1 現在の自分の技量で具現化可能かの見当も必要ですが、「できらぁ！」の前提のもとに詰めておくのが良いでしょう。駄目なら途中でその部分だけすっ飛ばし、次回作に回せばよいのですから。

さて、ここまで画面に表示させる様々なキャラクタなどを挙げてきたわけですが、いよいよ実際に「状態」とほぼ一致する所まで分解することにします。もちろん上に挙げた全てについて説明していくことは、誌面の問題もあって現実的ではありません。そこで簡単のために一番シンプルな挙動をする「自機の弾」を例に取り上げてみましょう。

では「自機の弾」に必要なのは、どのような処理になるでしょう？、ゲームのルールから考えてみます。

○プレイヤーの操作によって「自機」から発射される
○「自機」の方向（この場合は画面上方向）に向かって移動する
○「敵」にぶつかると消える
○移動した結果画面外に出たら消える

この中で「自機からの発射」と「敵にぶつかると消える」処理に関しては、「自機の弾」自身が行う処理ではないので除外できます。そのため、必要な「状態」は、最低限「画面上の座標」と「弾が生きているか否かのフラグ」があれば良いことになります。

つまり「自機の弾」が自分自身で持っていなければならない「状態」は最低限

(1) 弾（自分自身）の画面上の位置
(2) 弾（自分自身）の生死フラグ

の２つさえあればよいのです。

さて、いよいよこうしたルールに従った処理と「状態」を実際のコード…ここでは特定の開発環境を考えない擬似コードとして書き表してみましょう。

```
//  shx、shy は弾の画面上の座標
//  shFlag は弾の生死フラグ
//  vy は処理一回ごとの弾の y 方向の移動量
//  dRUy は表示画面の長方形（矩形）の四隅のうち、左上の y 座標

// 弾の移動処理。画面上方向にしか進まないという前提（vy は負の値）
shx = shx + vy;

// 画面からはみ出たら処理終了（上にしか進まないので y 座標の判定だけ行えばよい）
if(y < dRUy){shFlag = kLD_STANDBY;}
```

実質、たった２行のコードです。複雑な処理をしたいのなら、これを基に様々に書き加えていく必要があるのですが、基本的にはこれで十分なのです。

ここまで設計図を元にアイデアを「状態」として落とし込み、ルールと「状態」を擬似コードとして書きおろせるまでの説明をしてきました。

実際のところ「この程度かよ？！」と思われた方がほとんどだと思います。ところが、実際にゲームを組んだ経験が多くない方にとっては、「この程度」の事が実は教えてもらわないと気がつけない、自分ではなかなか辿り着けないノウハウの一つです。ゲームとは、こうした簡単でつまらない処理の積み重ねの「状態の入れ子構造」で構築されている、と言って良いでしょう。(※1)

■「状態の入れ子構造」

ここまで説明してきて気がついた方もいるかもしれませんが、ゲームにおける「状態の入れ子構造」は、いわゆるオブジェクト指向言語のクラスの概念と近いものです。ゲーム開発の関連書籍でもそうした説明を行い、可能な限りクラスを使うことを指示しているものも多く見られます。ですが「必ずクラスの概念を使わなければならない」という訳ではありません。そうした面倒くさい考え方をしなくても、もっと簡易な方法で実現することもできます。C 言語などの変数でも十分実現できますし、ウディタでも可能です。

要は、自分の手のつけられる範囲、今理解できている方法で実装していけばよいのです。

構造体やクラスを利用するのは、その言語に習熟しているか、アプリケーションの構築の経験をある程度積んでからでも遅くはありません。まずはアプリそのものを完成させることを目標にしましょう。

次の章では、いよいよ自分の想像するゲームアプリを完成させるために、どうしても必要となる「思いつきを形にする手順」について説明していきます。

※1「状態」を変数と言い換えてもいいでしょう。コーディングできるレベルまで分解するまでは「概念ですら代入することができる」魔法の変数ですが。

part2　思いつきを形にするために

■集めるだけでは形にならぬ

ここまで「ゲームは状態」であり、アイデアを「状態」に分解していけばゲームになる、という事を説明してきました。

しかし、それだけ理解できればゲームが完成するわけではありません。まず何かを思いつき、それをゲームの形にして、それをプログラムできるように落とし込む必要があります。

先程の言葉を使えば「思いつきを状態に書き換えていく」という事になるでしょう。

「そんなの分かるわけないじゃん」と言われる方も居るはずです。ですが、自分が思いついたことを「状態」に書き換えていくことが出来なければ、ゲームを完成させることはできません。

一方で、「ネットで拾ったソースとかをコピペしていけば、何かできそうだから、今はやらなくていいや」、と思う方も居るでしょう。たしかに最近は、様々な情報をネットで入手することが可能です。オープンソースの隆盛によって、自分の使う開発言語や環境で使うことのできる、「ゲームを構築していく時に必要になるはず」のソースコードを比較的楽にに入手することもできます。そして、ネットから得た情報を元に、ダウンロードしてきたソースを組み合わせていけば、そのうち自作ゲームが完成してしまうような気がするのも、仕方のないことです。

しかし、手に入れることのできた情報やそこ紹介されているサンプルなどを実際に動かし、関数やメソッドなどの内部の働きに納得したり、様々なテクニックに驚くことはできたとしても、それでその知識を活用できるようになった訳ではありません。手に入れたその断片的な知識を、自分の作ったプログラムに活用できるようになるまでには、相応の時間と苦労が必要になります。絵の How to を集めても、実際に描かないことには絵が上手くならないのと同じです。無数の「いつか必ず役に立つはずの情報」を手に入れたとしても、それをまったく利用できないままプログラムに対する意欲を失って、何も完成させることなく終わってしまう、というのは珍しいことではないのです。

■どこまで想像してますか？

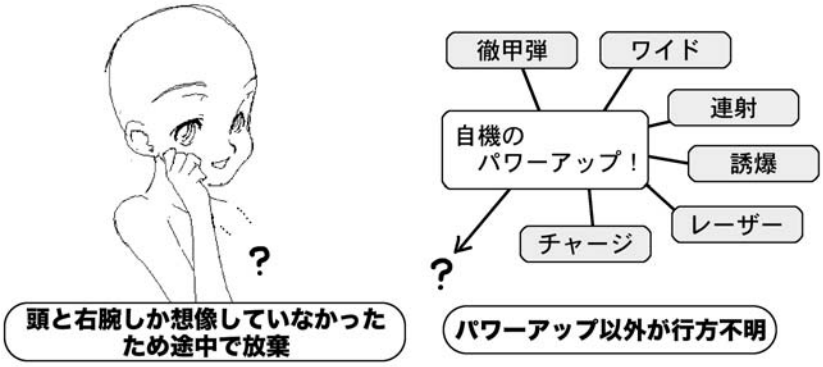
プログラムに不慣れな人だけでなく、ある程度技量の上がった人でもそうなのですが、アプリ構築に挫折してしまう原因のほとんどは、実は「自分が作りたいゲーム」の姿がはっきりしていない、という点にあります。

人間の脳は思ったよりもいい加減で、頭の中だけで何かを考えているとき、考えているものの全体像が完成していなくても、全てハッキリしていると思い込んでしまうのです。これは、必要な要素のみを明確に、それ以外を省略して覚える脳の記憶システムに起因しており、自覚していないと全く気が付かないままになってしまうトラップの一つです(※1)（fig.1-4）。

そのため、自分の頭の中の完成図だけを頼りに完成を目指すと、多くの場合途中で破綻して、

※1 だからといって脳の記憶システムが欠陥だというわけではありません。生命の進化の中で、すばやく記憶し、必要に応じて思い出すためにこうしたシステムが必要になったとのでしょう

Fig. 2-1　イメージできないないところは具現化出来ない



完成させることができなくなってしまうのです。

これは、自作絵を描く時に普段描いているポーズやアングルと違うものを描こうとすると、突然描けなくなるという場合によく似ています。

こうした事を防ぐためには、一つの方法と、それをより役立てるための一つの注意があります。

その方法とは、先に説明した「全ては状態が支配する」で説明したとおり、頭の中だけの完成図をパソコンのメモ帳などではなく『紙』に書き出すことです。

自分の考えを紙に書き出すことで、自分が何を考えているのか、どこをきちんと考えていて、どこを考えていないのか、を明確にするのです。これを行うことで、頭の中だけで考えていて実現できそうもないアイデアに囚われて新しい発想ができなくなったり、収拾の付かないモヤモヤとしたアイデアだけが漂うような状態に陥ってしまう事を防ぐことができます。

また、紙に書いておくことで、やろうと思っていたアイデアを忘れたり、自分が作りたかったものを見失うことも防ぐとこができますし、脳内だけで描いた設計図の勘違いや見落としに気づかないまま作業をすすめる事を防ぐこともできます。

最初はもちろん単なる落描きメモ程度であっても十分です。自分の持っているアイデア､イメージ､「こんな感じ」といった「感覚」は、必ず紙に書き出し、可能であればそれについて人と話すなどして、明確にし、検討して、そこからまた考えるという癖を付けると良いでしょう。

次に注意するのは、高い位置に目標を置くのではなく、いまの段階で可能な範囲での完成を目指すことです。最初に「(市販ゲームの）○○みたいなゲームを完成させよう」といった、と唯一の高い目標を置くのではなく、目標をいくつもの段階に分けて「最初はウィンドウを表示させるだけ」､「次にキャラを表示する」などの形で､「最低限ここまで」「この部分を達成したら、次はここまで」という、仮のゴールをいくつも作っていくのです。

こうすることで、脳内で肥大化する一方の、手の付けられない大規模で無謀なプロジェクトに挑むことを防げます。また、最終的なゴールに辿り着くまでに、自分が考えている以上の時間は掛かったとしても、一つ一つの達成感を味わうことでプログラムに対する意欲を維持することができます。成果の出ない長い道のりで時間を費やしてしまうようなリスクを避け、着実に目標に辿り着く確率を高くしていけるのです。

※１ごく希に、初めてプログラムを始めた人が、比較的大規模なゲームを完成させてしまうことがあります。しかし、こうした人は例外的で、自分も同じ事ができるとは思ってはいけません。