■ Foreword

Thank you for picking up the "Kirikiri/KAGEX Course ~Digital Novel Creation~". In this book, I will explain the scripting for visual novel games using Kirikiri/KAGEX. Recent visual novel games feature highly advanced presentations and systems, which is making the scripting more difficult. Kirikiri/KAGEX allows you to achieve these things relatively easily, but the drawback is that the manuals are almost non-existent or unhelpful. I hope this book can improve that situation even a little bit.

When it comes to visual novel game creation tools, traditional ones like Kirikiri and NScripter are well known. Recently, various other tools have emerged, such as YU-RIS and CatSystem2, as well as Ren'Py from overseas. The focus of this book is Kirikiri, but instead of KAG—which is often covered in commercially available guidebooks—we will be using a system called KAGEX. KAGEX is a modified and expanded version of KAG that allows you to create modern-style visual novels relatively easily. I'll leave the details for the main text. It has become truly convenient compared to KAG, and even those who have used KAG before might be surprised.

The creation of visual novel games itself seems to be losing momentum compared to a while ago, which is a bit sad. I haven't been able to check it out yet, but I hear TYPE-MOON's new release “Witch on the Holy Night” is amazing, so I find myself wondering if the number of Kirikiri users might increase again. However, if you download Kirikiri and actually try using KAG, you might end up giving up due to the sheer gap in expectations. That's where KAGEX comes in, and by extension, this book.

Please allow me to take this opportunity to briefly introduce myself. My name is Sakano, and I am active in a personal doujin circle called Biscrat. Although I am still a university student, I have made a few doujin games. I usually mainly take on scripting work for visual novel game production. I am always open to job offers, so if you have any opportunities, please reach out to me using the contact information at the end of the book.

I have made a few visual novel games, albeit in a somewhat haphazard manner. I have tried to reflect those experiences in this book as much as possible. Since this is strictly from the perspective of a single programmer involved in visual novel game production, I imagine there might be some bias in the content. Completing a visual novel requires various elements beyond just scripts, such as scenarios and graphics. Making a game takes a lot of time and effort and is truly difficult, but it also has its own kind of fun. I would be happy if this book could be of some help in your creative activities. I am looking forward to playing the games you create.

2011 / 12 / 30　　sakano

Index

1. Introduction

This is about what Kirikiri / KAGEX, which this book explains, actually is in the first place. It is a somewhat trivial topic, so I hope you can just relax and read through it.

■ Kirikiri is a freeware that lets you make various things

Kirikiri is free, open-source software that allows you to create mainly Multimedia titles by writing in a scripting language called TJS2. (From the official Kirikiri page)

There might be some unfamiliar words mixed in, but in short, "Kirikiri" is free Software that can do various things. You can create software that outputs text, graphics, sound,and more on a computer. Since it is free, you can use it without paying.

A massive amount of games across various genres such as novel, shooting, role-playing, simulation, and action, as well as non-game applications, have been created. In particular, it is quite heavily used even commercially for the production of 18+ erotic games. To give some examples, games like "Fate/stay night" and "Fate/hollow ataraxia" by "TYPE-MOON", and "STEINS;GATE 8bit" by "Nitroplus" are games produced using Kirikiri.

The core of Kirikiri itself is made using a programming language called C++. Since it is developed as open-source, if you can use C++, you can even modify the core system. However, since this book is aimed at beginners, we won't be able to touch on this level of discussion at all. Being able to handle C++ will greatly expand the range of what you can do, so if you are interested, please give it a try.

The current version of Kirikiri is Kirikiri "2", but it seems the next version, Kirikiri 3, is also under development. The best source of information about Kirikiri 3 is the Twitter account of Kirikiri's creator, W.Dee (https://twitter.com/_w_dee). However, he almost never talks about Kirikiri. It seems that Kirikiri 3 is planned to include 3D features and operate on platforms other than Windows. It supposedly will also support smartphones like iPhone and Android. Development doesn't seem to be progressing much, but I'm looking forward to it.

■ TJS is the scripting language used in Kirikiri

To make something with Kirikiri, you need to use a scripting language called "TJS". TJS is apparently an acronym for Tiny Java Script. TJS is simpler when compared to C++,

but it is too complex for someone with no programming experience to use. Also, because TJS is almost exclusively a language for Kirikiri, there is little information available, and it is hard to say that there is a good environment for studying it on your own. When studying TJS, I recommend studying programming with a language other than TJS, such as JavaScript. There are many books for beginners, and a lot of explanations are available on the web. If you study TJS after gaining programming knowledge, it won't be that difficult. It may seem like a detour at first glance, but I think it will ultimately save time.

■ KAG is a system for making adventure games with Kirikiri

When you download Kirikiri from the official site, you will find that the "KAG" system is bundled with it. KAG stands for Kirikiri Adventure Game, and it is a system that runs on Kirikiri for creating adventure games.

As mentioned earlier, using TJS requires programming knowledge. Therefore, KAG has been prepared so that even people who are not interested in programming can create adventure games. The KAG system itself is built with TJS so that it can run on Kirikiri, but if you are just using KAG to make games, you do not need any knowledge of TJS. You can create an adventure game using only "KAG Script," which is simple enough for even non-programmers to use.

However, because KAG is a system created many years ago, its features are too lacking to create modern adventure games. If you work hard to customize KAG yourself, you can add most features, but doing so will still require knowledge of TJS. This is where KAGEX comes in.

■ KAGEX is a system modified from KAG

"KAGEX" is a system for creating novel games, developed by Tsuyoshi Watanabe (https://twitter.com/wtnbgo) based on KAG. It is built as an extension of the aforementioned KAG. Since it is a system developed by a company that actually develops commercial games, various features are available right from the start. If it is a standard-format novel game, you can create it efficiently by writing "KAGEX Script". This book will introduce how to use this KAGEX.

While you can find explanations of Kirikiri / KAG on the web and in books, explanations of KAGEX are almost non-existent. Since there are considerable differences between KAG and KAGEX, please be careful when using KAG explanations as a reference.

■ Scripting languages and programming languages

This is a minor point, but it would be unhelpful not to explain it, so I'll write it down. At present, computers

have not reached a level where they can think and act automatically, so humans must create instructions for them in advance. Computers follow these instructions and operate exactly according to them. These instructions are called a "program".

And the process of creating a program is called "programming". When programming, a "programming language" is used. For example, there are a massive number of them, such as C++, Java, Lisp, Fortran, and Prolog. A "scripting language" is a type of programming language, and it refers to a relatively simple programming language. Examples of this include JavaScript, Lua, and Awk, which again exist in massive numbers. TJS, which is used in Kirikiri, is also a type of scripting language. As an aside, in Kirikiri, you can also use scripting languages other than TJS, such as JavaScript and Squirrel. If you want to use them, it might be a good idea to look into it.

The main content of this book is an explanation of how to write the scripting language used in KAGEX, called KAGEX Script.

■ Adventure games and visual novel games

I explained that KAG is a system for creating "adventure games," and KAGEX is a system for creating "novel games". What exactly is the difference between an adventure game and a novel game? I suspect that there probably isn't a widely shared definition.

Personally, I consider adventure games to be those with command-input or command-selection systems, like "The Portopia Serial Murder Case." When I think of novel games, they are games where choices are occasionally interspersed between parts of the story, which make up the majority of recent 18+ novel games. When I ask others, it seems they also call the latter "adventure games," so it appears I am the one who is mistaken.

There also seems to be a classification where full-screen text display is an adventure game, and display on the bottom half of the screen is a novel game, so I think it's really inconsistent depending on the person. Or again, some people say that things like novels aren't games at all, and in fact, there are linear novels where the player cannot intervene at all and just watches.

Discussing "gameplay" and such is a heavy burden, so I will omit it. Creating games is fun, but researching games is also surprisingly interesting, so why not give it a little thought?

To get back on track, since I still hold the personal feeling that adventure games have a higher degree of freedom than novel games, I am treating KAGEX as being for novel games. Although it would be fine to say both are for adventure, since they are both Kirikiri Adventure Games.

KAG does not strictly limit its applications, so if you push it, you can do various things using just KAG scripts. Compared to that, KAGEX is a system designed specifically for producing games in the format common to 18+ visual novels. The types of games you can make are more limited than with KAG, but specific types of games can be produced efficiently. Well, if you use TJS, you can still do a lot with it, though.

It only has the vague meaning that KAGEX seems to have narrower applications than KAG, so please don't worry about it.

■ About "digital" novels

By the way, in the title of this book, I use the term "Digital Novel" rather than "Adventure" or "Novel." I define a Digital Novel as digital content that conveys a story primarily through text expression, encompassing both adventure games and visual novels. As usual, this is my own personal definition that doesn't really translate to anyone else. From here on, I will not use words like "novel" or "adventure," but will consistently use the unified term "Digital Novel."

I think that people who like Digital Novels simply love reading stories. There are many other media besides Digital Novels that convey stories. Novels, manga, anime, movies—the list is endless. Why, then, choose Digital Novels from among them? Is it not a novel that you actually want to create, or perhaps a manga? Thinking about this before you start production might save you from wasting your time.

Compared to other genres of games, Digital Novels can be created with less effort. They are produced even within the scope of hobbies like doujin activities, and diverse works are being created. That said, since there are many materials to prepare such as text, sound, and graphics, it requires a great deal of hard work from multiple people. If you are trying to make something proper, you had better be prepared. I recommend that absolute beginners try completing one title alone before involving others. It is fine to use free materials found on the internet for art and sound. Trying to gather members to create a masterpiece from the start is a recipe for failure. A new page will be added to your dark history.

While producing a Digital Novel is that difficult, let's think about the good points too. Compared to novels and manga, there is the advantage of being able to use non-text elements like graphics and sound as a means of expression. This is true for manga as well, but art becomes part of the expression, not just text. Beautiful BGM and voice acting enhance the scenario and stir the player's emotions. Also, compared to movies and anime, the fact that you can use text

is an advantage. You don't need to prepare graphics for every single scene frame by frame; expressing it in text is sufficient. You can also introduce novel-writing techniques into the text.

As a humble programmer, what I particularly want to say is that in Digital Novels, the system itself can become part of the expression. Since it is a game running on a computer, all forms of expression achievable on a computer are available for use.

"Choices" are a de facto standard game system widely adopted in Digital Novels. They are by no means merely for letting the player choose a route.

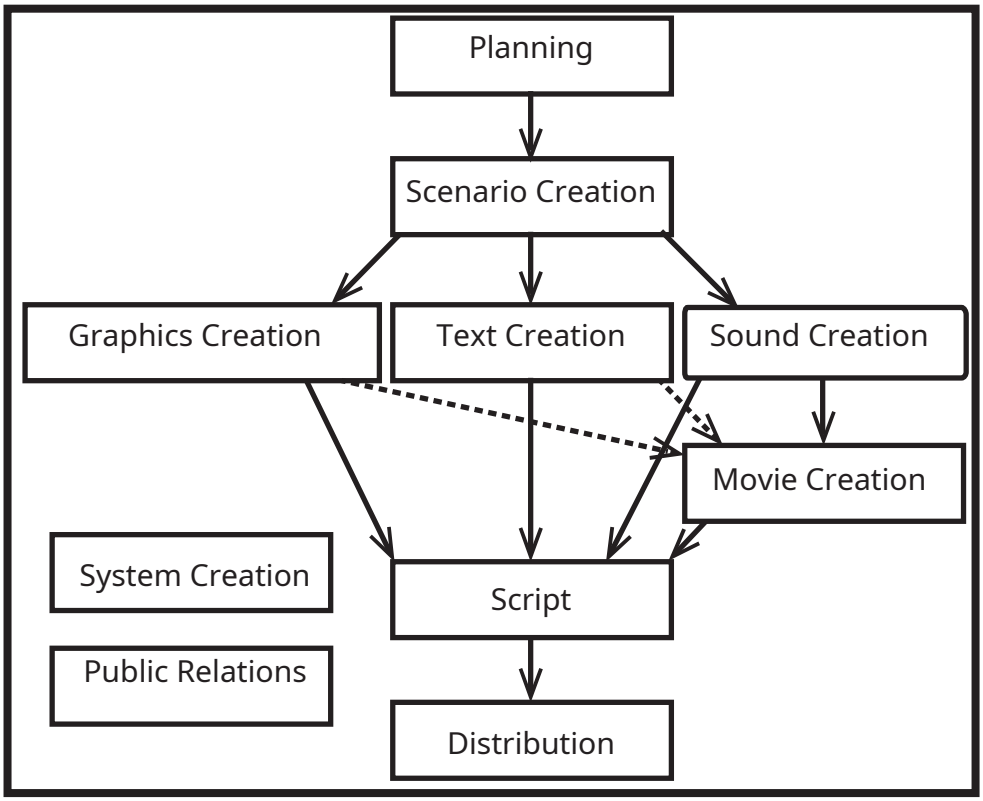
As an example, let me introduce 07th Expansion's "Umineko: When They Cry." It spans from Episode 1 to 8, and since each one is long, it isn't exactly a game I can recommend to just anyone... However, at the climax of the final chapter, Episode 8, a question related to the story's theme is posed in the form of a choice. The player, who until then has merely been reading passively, is forced to stop, think actively, and make a decision. This left a very strong impression on me, as I had just been reading along idly.

I will mention one more example of expression through game systems: Key's "Little Busters!". This game falls into the so-called "school life" genre, and it features a batting practice minigame that takes place every day after school as a club activity. Just seeing the cute pixel art move around during the minigame helps build affection for the characters. But it doesn't stop there; the minigame itself replaces textual narration like "everyone practiced together." It becomes a key element that also visually depicts the process of gathering members and signifies the characters' growth.

If you are a writer, I would be happy if you could consider for a moment whether something really needs to be expressed in text. In that respect, the work arrangement where scenarios are paid by the kilobyte is not ideal. It would be best if a dedicated director were attached to oversee the scenario as a whole, though. In the world of doujin, I think there is a lot of flexibility. I look forward to seeing interesting games being created.

■ The flow of digital novel production

The diagram on the next page shows the rough workflow of Digital Novel production. This book explains the "Script" part. For the other parts, I will only provide a casual explanation here. Please figure out the details yourself by reading specialized books on each subject or something similar. In reality, even if you rely on others to do the work, you will still need a minimum level of knowledge to make the request.



It follows a waterfall model, stepping through stages in order from the beginning. However, as is the case with ideals versus reality, things actually never proceed this way. It is completely normal to write scripts even when all the materials are not ready. It's no use waiting around for them to be done, so it's a bit of a helpless situation.

• Planning

Think about what kind of game to make. If the genre is not a Digital Novel, then it's goodbye to this book. A planning document is important even from the perspective of an outsourcer. If it looks like an interesting game, since we are human after all, we naturally get more motivated than usual.

• Scenario Creation

In the case of a Digital Novel, if there is no story (scenario) at the core of the content, nothing can be done. It seems plots are also written to determine necessary materials like graphics and sounds. The stages of planning, scenario creation, and text creation are often particularly undifferentiated.

• Text Creation

If the text used in the game is not written with the game screen somewhat in mind, you will run into trouble during the scripting stage. If you haven't thought about it, the text might overflow from its frame,

or something similar might happen. At this stage, some directing instructions are also included to a certain extent. This involves things like which background or character sprite to display, or instructions for the voice actors. For now, it's best to be careful because novel text and game text are completely different things.

• Graphics Creation

In Digital Novels, the style of layering character sprite images on top of a background image and displaying text over that has become the standard. At key points, event illustrations (also called full-screen images, CGs, stills, etc.) are inserted. In addition to that, system images are also required, such as images used for the title screen and save screen, and the frame for the text display area. Smaller items include the game logo and the icon for the game's executable file. If you use Kirikiri, you will ultimately convert them into a proprietary file format called TLG. File formats will be explained in Chapter 3.

• Sound Creation

BGM, sound effects, system sounds (like the sound when a button is pressed), and so on will be necessary. Depending on the situation, you might record voices from voice actors, or record a theme song. In Kirikiri, you can use either wav files or ogg files. mp3 is a commonly used sound file format, but it is not used in Digital Novel production due to licensing issues.

• Movie Creation

This is necessary if you use things like an opening movie. Kirikiri can play MPEG1, WMV, and FLASH.

• Script

We will integrate the created assets to form a single game. This is the main focus of this book. We will explain this from the next chapter onwards.

• Distribution

Whether to offer it for download on a webpage, burn it to CD and distribute it at an event, sell it on consignment in a shop, or various other things. Feel free to do as you please.

• System Creation

The basic parts are handled by Kirikiri / KAGEX. This becomes especially necessary when a specific game needs unique features or includes minigames. In Kirikiri, programming is done using TJS or C++.

• Public Relations

Games whose existence is unknown will not be played. It is important. Webpages, blogs, Twitter, Pixiv, Facebook, news sites, events, magazines, flyers, and more. There seem to be various ways.

2. 開発環境の準備

この章では吉里吉里 /KAGEX の SDK など必要なファイルをダウンロードし、ゲーム製作ができる環境を整えます。

■拡張子の表示

拡張子が表示されていない環境の場合、これからの説明がわかりづらくなってしまうので前もって表示しておきます。既に表示されている環境の方は読み飛ばして構いません。取りあえずの説明はしますが、「拡張子」という言葉を初めて聞いたという方にはこの本の説明は難しいかもしれません。どうにも理解できなくなったらインターネットで検索でもして頑張ってください。

Windows では、「拡張子」というものでファイルの形式（種類）を識別しています。拡張子はファイル名の最後に「 . 」（ピリオド）で区切られてくっついています。例えば、「メモ .txt 」というファイル名だった場合、「 txt 」が拡張子ということになります。「 txt はメモ帳などで開くことができる「テキストファイル」の拡張子になります。他にも「 exe 」「 zip 」「 jpg 」など様々な拡張子が存在します。

ゲーム開発をする上で拡張子は重要になりますが、 Windows では拡張子の表示が省略されるという残念な初期設定になっています。これを省略されない設定に変更します。ここでは Windows 7 の場合の変更方法を説明します。 XP や Vista などの場合は、「 windows xp 拡張子 表示」などで検索すると、画像付きでわかりやすいウェブページがいくらでも出てくるので参考にしてください。

まず、左下のスタートメニューから「コントロールパネル」をクリックします。コントロールパネルが開くので、「デスクトップのカスタマイズ」→「フォルダーオプション」の順にクリックします。次に、「表示」タブを選択して「詳細設定」の中から「登録されている拡張子は表示しない」を探しだして、チェックを外します。「 OK 」ボタンをクリックすれば拡張子が省略されないように設定が変更されます。

■吉里吉里 /KAGEX の SDK を入手

吉里吉里 /KAGEX でゲームを開発するには、吉里吉里本体の他にも KAGEX やその他のプラグインなど、色々と必要なファイルがあります。ひとつずつダウンロードしてくるのは面倒なので本書のサポートページでまとめて SDK(Software Develolpment KIt, ソフトウェア開発キット) としてダウンロードできるようにしてあります。「 http://biscrat.com/book/kagex/support/ 」にアクセスしてダウンロードして下さい。

手に入れた SDK は ZIP 形式で圧縮されています。各自で解凍してください。わか

らない方は次節の「圧縮 / 解凍」を参照してください。解凍すると「 KAGEX_SDK 」のようなフォルダができると思います。「 readme.txt 」が入っているので、クリックして読んでおいてください。

「 KAGEX_SDK 」フォルダ内の「吉里吉里安定版 SDK 」フォルダにも「 readme.txt 」と「 license.txt 」が入っています。こちらもしっかり読んでおいてください。吉里吉里を使用する際のライセンス（利用許諾条件）になります。 W.Dee 氏が製作した吉里吉里を使わせてもらうことになります。もちろん好き勝手に使う訳にはいかないので、そこに書かれている条件は必ず守らなければなりません。以下の説明は全く正式なものではなく、理解の助けになるように、かなりテキトーに説明したものです。必ず「 license.txt 」を自分で熟読して理解してください。

吉里吉里は「 GNU GPL 」または「吉里吉里独自のライセンス」のどちらかのライセンスを選んで利用することができます。ゲームを作る上では「吉里吉里独自のライセンス」の方が重要です。 GPL の方で吉里吉里を利用している例は見たことがないです。「吉里吉里独自のライセンス」では吉里吉里を無償で利用することができます。製作したゲームを無料配布する場合も、お金を取って配布する場合も変わりません。ゲームとして配布する際に「吉里吉里を使っています」などと表示する義務もありません。すごい太っ腹です。 C++ を使って吉里吉里本体を改造する場合は他にも条件がありますが、本書ではそんなところまで扱いません。

「吉里吉里安定版 SDK 」フォルダには、ライセンスの他に「 kag3 」フォルダと「 kirikiri フォルダが入っています。

「 kag3 」フォルダには KAG システムとその説明書などが入っています。本書では KAG ではなく KAGEX を利用するので kag3 フォルダは使用しません。説明書などを読んでみてもいいですが、 KAGEX と全く異なることは意識しておいてください。下手に読むと混乱します。

「 kirikiri2 」フォルダには吉里吉里の本体と説明書が入っています「 krkr.eXe 」が吉里吉里本体になります。クリックしてみると「フォルダ / アーカイブの選択」というウィンドウが出てきます。まだ準備が済んでいないので取りあえず「キャンセル」をクリックして終了しておきます。「 kr2doc 」フォルダに吉里吉里本体の説明書 (リファレンス) 、「 tjs2doc 」フォルダに吉里吉里で使用する TJS の説明書が入っています。いきなり読んでも難しいですが、慣れてくるとこれらも使うようになるかもしれません。説明書の存在は頭の片隅にでも置いておくといいです。「 tools 」フォルダにはゲーム製作をする上で使用するツールが入っています。本書でも一部のツールを利用します。

■圧縮 / 解凍

「圧縮」とは、ファイルのサイズを小さくすることです。サイズが大きいとインターネットを通して送受信するのに時間がかかってしまうので、インターネットで配布されているファイルの大部分は圧縮されています。圧縮の形式には ZIP 以外にも LZH 、RAR 、 CAB 、 7Z など、様々なものがあります。 ZIP 形式は最もメジャーな圧縮形式になります。

圧縮されたファイルは直接使えません。圧縮されたファイルを圧縮する前の状態に戻すことを「解凍」といいます。吉里吉里を使うには ZIP 形式で圧縮されたファイルを解凍する必要がありますが、解凍するには専用のアプリケーションが必要になります。ここでは有名な「 +Lhaca 」の使い方を紹介しておきます。

+Lhaca は「 http://park8.wakwak.com/~app/Lhaca/ 」からダウンロードできます。ダウンロードしたファイルを実行してインストールしてください。インストールが完了すると、デスクトップに +Lhaca のアイコンができていると思います。そのアイコンに、ダウンロードした「 kagex_sdk.zip 」をドラッグ＆ドロップすると、自動的に解凍され、デスクトップに「 kagex_sdk 」というファルダができます。それが解凍された吉里吉里のフォルダになります。解凍してしまえば解凍前の「 kagex_sdk.zip 」は不要なので削除しても構いません。

■素材ファイルの入手

スクリプトは本を読むだけで使えるようにはなりません。読みながら試せるように、本書のサンプルスクリプトで使用している画像や音などの仮素材をダウンロードできるようにしてあります。こちらも SDK と同様にサポートページからダウンロード、解凍しておいてください。

■開発用フォルダの準備

SDK には必要なファイルが揃っていますが、フォルダごとにバラバラに入っているため使いづらいので、開発用のフォルダを作成してファイルを配置します。

まずは開発用に新しいフォルダを作ります。フォルダ名は何でもいいですが、例としてデスクトップに「開発用フォルダ」という名前で作成しておきます。デスクトップでなくとも、自分で使いやすい場所ならどこでも構いません。作成したら、 SDK フォルダの「ツール」フォルダに入っている「 make_dev_folder.bat 」をダブルクリックで実行してください。「開発用フォルダの選択」というダイアログが開くので、先ほど作成したばかりの「開発用フォルダ」を選択して「 OK 」ボタンを押します。これで「開発用フォルダ」に使用するファイルが全てコピーされます。

「開発用フォルダ」にコピーされた「 krkr.exe 」を起動してみてください。ゲーム画面が無事に表示されれば OK です。

■.Net Framework 4 の準備

いくつかの開発ツールを使うには「 Microsoft .Net Framework4 」をインストールしておく必要があります。 Microsoft のページからダウンロード、インストールしておいてください。「 http://www.microsoft.com/downloads/ja-jp/details.aspx?displaylang=ja&FamilyID=9cfb2d51-5ff4-4491-b0e5-b386f32c0992 」からダウンロードできます。

■テキストエディタの準備

スクリプトを書くには、「テキストエディタ」という種類のソフトウェアが必要です。自分で使い慣れたものがあれば、それを使ってください。注意としては、『 Microsoft Word 』や『一太郎』のような「ワープロソフト」では駄目です。ワープロソフトでは文字の他に画像を挿入したり、文字の大きさを変えたりという機能がありますが、スクリプトで使えるのは単純な文字のみです。そのため、文字だけを扱うのに特化したテキストエディタを使います。

「 Widows 標準のメモ帳でも大丈夫です」などと省略してしまいたいところですが、本当にメモ帳を使ってしまう人がいると悲しいので説明します。メモ帳とその他の高機能なテキストエディタでは効率が段違いになります。私が使っているのは『サクラエディタ』や『 Emacs 』というテキストエディタになります。ここではサクラエディタの導入の仕方を紹介します。詳しい使い方などはサクラエディタのヘルプを参照してください。

まずは必要なファイルをダウンロードしてきます。サクラエディタのウェブページ「 http://sourceforge.net/projects/sakura-editor/ 」から「 Download 」をクリックするとダウンロードが始まります。「 sakura-2-0-3-1.zip 」のような ZIP 形式ファイルになっているので解凍してください。解凍すると「 QuickStartV2.zip 」と「 sakura.exe 」が入っています。「 QuickStartV2.zip 」は不要なので削除して構いません。「 sakura.exe 」の方「サクラエディタ」などのフォルダを新規作成してその中に入れておきます。

次に、『 bregonig.dll 』のウェブページ「 http://homepage3.nifty.com/k-takata/mysoft/bregonig.html 」から Unicode 対応版の「 bron205.zip 」のようなファイルをダウンロードします。番号の部分は変わっているかもしれません。こちらも ZIP 形式ですので解凍してください。解凍すると「 bregonig.dll 」というファイルがあるので、先ほどの「 sakura.exe 」と同じフォルダに移動してください。その他のファイルは不要なので削除して構いません。

最後に、本書のサポートページ「 http://biscrat.com/kagex/support/ 」からサクラエディタ用の吉里吉里向け設定ファイルをダウンロードします。 zip 形式になっているので解凍し、出てきた「 krkr_sakura 」フォルダを「 sakura.exe 」と同じ場所に移動します。

ここまでできたら、「 sakura.exe 」をクリックして実行してください。吉里吉里用に設定します。メニューの「設定」→「タイプ別設定一覧」→適当に使われていない設定を選択（「設定 20 」など）→「インポート」→「 krkr_sample 」フォルダの「 tjs.ini 」を選択して「開く」→「はい」→「 OK 」→「 OK 」。以上で TJS スクリプト編集用の設定ができます。同様に、設定 21 あたりに「インポート」から「 krkr_sample 」フォルダの「 ks.ini 」もインポートしてください。

最後に、 KAGEX スクリプトファイル（拡張子が ks のファイル）と、 TJS スクリプトファイル（拡張子が tjs のファイル）を自動的にサクラエディタで開けるように設定します。このように拡張子ごとに使用するソフトウェアを指定することを「関連付け」と言います。文字だけでわからなければ検索するとわかりやすい解説が出てきます

「開発用フォルダ」の中の「 data 」フォルダの中に「 startup.tjs 」があります。ダブルクリックすると、「このファイルを開けません」のようなダイアログが出てくるので、「インストールされたプログラムの一覧からプログラムを選択する」にチェックを入れ、「 OK 」ボタンをクリックします。「ファイルを開くプログラムの選択」ダイアログが表示されたら、「参照」ボタンから「 sakura.exe 」を選択して「開く」ボタンを選択します。「この種類のファイルを開く時は「選択したプログラムをいつも使う」にチェックが入っていることを確認し、「 OK 」ボタンを押すと、サクラエディタで「 startup.tjs 」が編集できる状態になります。一度サクラエディタを終了して、「 startup.tjs 」をダブルクリックするとサクラエディタが実行されることを確認してください。同様に、「 scenario 」フォルダに入っている「 01.ks 」もサクラエディタに関連付けします。やり方は全く同じです。以上で、拡張子が tjs のファイルと ks のファイルをサクラエディタで編集できるようになりました。

これはお好みですが、「設定」メニューから「タブバーを表示」をクリックすると、複数のファイルを編集する際にタブが使えるようになります。便利なので設定しておくことをお勧めします。その他にも様々な設定項目があります。サクラエディタのヘルプも見つつ自分が使いやすいように設定していくと効率が良くなっていきます。

3. スクリプトの基本

この章からいよいよ KAGEX スクリプトを書きはじめます。文字や画像の表示、音の再生など、基本的な機能をひと通り触ってみます。

■フォルダ構成

ゲーム開発中は「開発用フォルダ」の「 data 」フォルダの中身をいじっていくことになります。この data フォルダは「プロジェクトフォルダ」とも呼びます。ゲームに使用する素材やスクリプトファイルは全てこのプロジェクトフォルダに入れることになります。

そのプロジェクトフォルダの中にもたくさんのフォルダが入っています。ここで少し説明しておきます。

素材ファイルを入れるフォルダ一覧	
bgimage	背景画像を入れます。
fgimage	立ち絵画像を入れます。
evimage	イベント絵画像を入れます。
uiimage	ボタンなどのインターフェース画像を入れます。
rule	トランジションのルール画像を入れます。
thum	サムネイル用の画像を入れます。
image	その他の画像を入れます。
bgm	BGM 用の音ファイルを入れます。
sound	効果音用の音ファイルを入れます。
voice	ボイス用の音ファイルを入れます。
video	ムービーファイルを入れます。
other	その他のファイルを入れます。

スクリプトファイルを入れるフォルダ一覧	
system	KAGEX システムの TJS スクリプトが入っています。
biscrat	著書が書いた KAGEX を拡張する TJS スクリプトが入っています。
main	その他の TJS スクリプトを入れます。
init	KAGEX の設定を変更するためのファイルが入っています。
sysscn	システム関連の KAGEX スクリプトが入っています。
scenario	ゲーム本編の KAGEX スクリプトが入っています。

素材ファイルを入れるフォルダ一覧は書かれている通りです。全て 1 つのフォルダに入れてしまうよりは分別した方がわかりやすくなります。製作ツールもフォルダごとに区別して動作しているので、素材を入れる先のフォルダは間違えないでください。

スクリプトファイルを入れるフォルダの方で、 system 、 biscrat 、 main フォルダについてはこの本では触れません。次のステップアップとしてこのフォルダの中身も使えるようになると、様々なことができるようになってきます。興味のある方は挑戦してみてください。 init 、 sysscn 、 scenario フォルダの中身がこの本の主眼となります。

「 data 」フォルダ直下の「 startup.tjs 」というファイルは、吉里吉里が実行されると一番最初に実行されるスクリプトです。編集することはないので気にしないでください。その他の「 default.rpf 」などのファイルは、各種ツールの設定ファイルになります。直接使用することは有りません。

■テキストの表示

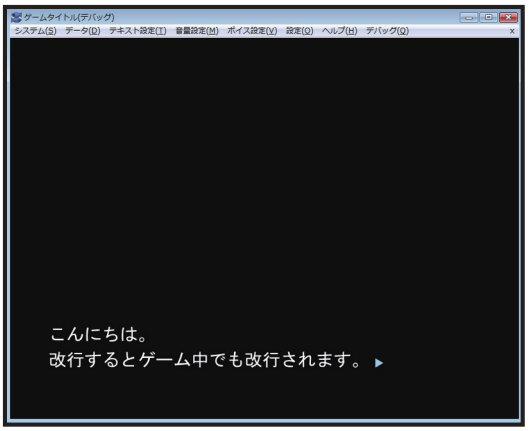
長い前説でしたが、ようやく KAGEX スクリプトの解説を始めます。まずはテキストを表示してみます。ゲーム本編のスクリプトは「 scenario 」フォルダの「 01.ks 」から始まります。テキストエディタで「 01.ks 」を開いてください。最初は以下のようなスクリプトが書かれています。

```
*start| スタート
こんにちは。
```

最初の行の *start| スタート はラベルというものです。ラベルについては後ほど説明するのでとりあえず無視してください。次の行には こんにちは。と書かれています。これだけでゲーム内で「こんにちは。」と表示することができます。試しに「 krkr.exe 」を起動すると「こんにちは。」と表示されます。

```
*start| スタート
こんにちは。
改行するとゲーム中でも改行されます。

空行を挟んだのでゲーム中でも改行されます。
```

テキストはただ打ち込むだけでゲーム中で表示することができます。以上のスクリプトを 01.ks に入力、保存して、 krkr.exe を起動してみてください。改行もゲーム中に反映されます。また、何も書かれていない空行を挟むとクリック待ちとページ送り(表示されているテキストの消去) ができます。



```
*start| スタート
地の文です。

【しおり】「こんにちは。
キャラクタのセリフです。」

【しおり】
「セリフを次の行に書くこともできます。
セリフ中の改行は、カギ括弧に合わせて自動的にインデントされます。」
```

【】でキャラクタの名前を囲むと、発言したキャラクタの名前が表示されます。キャラクタ名は通常のテキストが表示される場所とは異なる、専用の名前欄に表示されます。【しおり】のあとに改行を挟み、次の行からセリフを書く場合は、「】」の後にスペースなどが混じらないように注意してください。

```
*start| スタート
【しおり /????】「名前欄には???と表示されます。」
```

シナリオ展開上でまだ名前が不明な場合など、キャラクタの実際の名前と異なるテキストを名前欄に表示したい場合は、「 / 」（半角スラッシュ）を挟みます。単純に「【???】とすると、テキストの表示は大丈夫ですがボイスを再生するときに困ります。詳しくは 5 章のボイス再生のところで説明します。

```
*start| スタート
【しおり /????】「[ 漢字 ' かんじ ] などに [ 振仮名 ' ふりがな ] を [ 振 ' ふ
```

ルビ（ふりがな）を振ることもできます。ルビを振りたい部分を [] で囲み、 '(半角アポストロフィ) の後ろにルビとして表示する文字を書きます。

テキストの基本はこのくらいです。文字の色や大きさを変えたりなど、細かい機能は後の章で紹介します。

■吉里吉里の実行について

「開発用フォルダ」の「 krkr.exe 」をダブルクリックすると吉里吉里が実行できます。また、「 exec_release.bat 」や「 exec_debug.bat 」をダブルクリックしても実行できます exec_release.bat では、吉里吉里をデバッグモードオフで実行します。開発中はデバッグモードで実行した方が便利なので、こちらはほとんど使用しません。 exec_debug.bat ではデバッグモードオンで実行できます。開発中はこちらを使用してください。

また、スクリプトの動作を確認するたびに exec_debug.bat をクリックするのは手間がかかって大変なので、テキストエディタのショートカットに登録しておくと便利

です。大抵のエディタには外部プログラムをショートカットに登録できる機能があると思います。

以下、サクラエディタの場合のやり方を書いておきます。詳しくはサクラエディタのヘルプを参照してください。少し面倒ですがこれから何十回と吉里吉里を起動することを思えば安いものです。他のエディタを使っている場合でもやっておいてください。

サクラエディタを起動したら、「ツール」メニューの「キーマクロの記録を開始」をクリックします。次に、同じく「ツール」メニューの「外部コマンド実行」をクリック、「 ... 」ボタンから「 exec_debug.bat 」を選択し、「実行」ボタンをクリックします。吉里吉里が起動しますが、今は使わないので終了しておきます。サクラエディタに戻り、再度「ツール」メニューの「キーマクロの記録終了＆保存」をクリックすると、マクロとして保存できます。サクラエディタ本体と同じフォルダに「 macro 」フォルダを新規作成して、その中に「 krkr.mac 」という名前で保存します。

マクロが保存できたら、そのマクロをショートカットキーで実行できるようにします。「設定」メニューの「共通設定」をクリックします。「マクロ」タブを開き、「参照」ボタンから先ほど新規作成した「 macro 」フォルダを選択します。下の「 File 」の欄で「 krkr.mac 」を選び、名前欄に「吉里吉里起動」、 Id 欄を「 0 」にして「設定」ボタンをクリックします。次に、「キー割り当て」タブを開き、「種別」から「外部マクロ」を選択すると、「機能」欄に「吉里吉里起動」が出てくるので、それを選択します。後はショートカットとして登録するキーを選択しますが、ここでは例として (Ctrl+Q) で吉里吉里が起動できるように設定します。「 Ctrl 」のチェックボックスにチェックを入れ、「キー」欄から Ctrl+Q を選択します。この状態で「割付」ボタンを押せばショートカット登録は完了です。「 OK 」ボタンを押して共通設定ダイアログを閉じてください。登録ができたら、試しに Ctrl+Q を押すと吉里吉里が実行されることを確認してください。

■デバッグモードについて

デバッグモードで吉里吉里を実行すると、開発するのに便利な機能が使用できます。デバッグモードオンでは、ゲームウィンドウの下に「 KAGEX ログ」というウィンドウが出て、クリプトに間違いがあるときに知らせてくれたりします。邪魔かもしれませんが、開発中は表示しておくことを推奨します。このウィンドウを閉じてしまった場合は、吉里吉里の「デバッグ」メニューの「 KAGEX ログモード」をクリックすると表示できます。「デバッグ」メニューには他にもいくつか項目がありますが、いくつかの機能は後ほど紹介します。

■ショートカットキーについて

マウス操作は簡単ですが、キーボードでの操作に慣れると格段に作業のスピードが上がります。 Ctrl+S(Ctrl キーを押しながら S キーを押すことです) でファイルを保存、Ctrl+Z で直前の操作取り消し、などは非常によく使うショートカットです。 Ctrl+T で半角英数字、 Ctrl+U で平仮名、 Ctrl+I で全角カタカナ、 Ctrl+O で半角ｶﾀｶﾅ、 Ctrl+P で全角英数字に変換、などは知ってると地味に便利です。 Ctrl+M は Enter の代わりになります。紹介しきれませんが他にもたくさんあります。初期設定ではショートカットキーが無くとも、よく使う機能は自分で登録しておくといいです。カーソル移動なんかもショートカット登録したりします。 1 日何時間もキーボードに向かっているとキーボードの右側まで指を伸ばすのも面倒なのです。カーソルキーなど時間の無駄です。そこまでやるかは別として、ゲーム製作なんて時間がいくらあっても足りないので、少しでも楽をする努力はするべきだと思います。まあ短縮された時間はやはりゲーム製作に注がれる訳ですが……。

■画像を表示する

次は画像を表示してみます。まだ仮素材として使う画像をダウンロードしていない場合は、サポートページ（ http://biscrat.com/kagex/support/ ）からダウンロードしておいてください。

KAGEX で表示する画像は、背景、立ち絵、イベント絵、その他にわけられます。 (テキスト枠やボタン、スライダなどシステムで使用する画像は含んでいません。それらは 6 章で扱います。) それぞれ使い方が異なるので、これからひとつずつ説明していきます。その前に KAGEX の画像表示の基本的な部分を説明しておきます。

■伸ばし棒の省略

コンピュータ (コンピューターに非ず) 関連では単語末尾の伸ばし棒を省略することが多いです。レイヤーではなくレイヤ、スライダーではなくスライダです。大した意味は有りません。読みづらいかもしれませんがそうなってしまうので慣れてください。「プレイヤー」は例外になっています。プレイヤだと何となく違和感があったので伸ばしてます。ゲームとプレイヤーの関係というような、どちらかというと人文系の視点で見ているからかもしれません。プレーヤー? プレーヤ? などと悩んだ結果プレイヤーに統一しました。プレーヤだと mp3 プレーヤみたいな意味に見えますね。本当にどうでもいい無駄話でした。

■レイヤについて

画像は「レイヤ」という透明な板に貼り付けて表示されます。フォトショップなどの画像編集ソフトのレイヤと同じようなものです。細かいことは割愛しますが、レイヤには前後関係がある、というのが重要です。後ろのレイヤに表示されている画像は、

前のレイヤに表示されている画像の後ろに隠れます。レイヤは、「背景＜立ち絵とその他＜イベント絵」の順番で後ろから並んでいます。背景が一番後ろ、イベント絵が一番前に表示されます。立ち絵は同時に何人も表示されたりするので立ち絵の間にも前後関係があります。詳しくは後ほど説明します。これらのレイヤのさらに上にテキスト枠とテキストが表示されます。

■画像の形式について

画像の形式には様々ありますが、吉里吉里で扱えるのは BMP 、 JPEG 、 PNG 、 ERI 、 TLG5 、 TLG6 、 WMF 、 EMF になります。

画像の形式には大きくわけてベクタ画像とビットマップ画像の 2 種類があります。吉里吉里で使える形式の中では WMF と EMF がベクタ画像になりますが、デジタルノベルを作る上ではまず使いません。筆者もほとんど使ったことがないため解説もできないので割愛します。残りの BMP 、 JPEG 、 PNG 、 TLG5 、 TLG6 はビットマップ画像になります。

BMP 形式はファイルサイズが大きくなりすぎるので基本的に使いません。 ERI 形式というのはよく知りません。使ったこともないので無視します。

JPEG 形式と PNG 形式はそこそこ使ったりします。両者の違いは、 JPEG が非可逆圧縮、 PNG が可逆圧縮であるということです。非可逆圧縮では、画像がボケたりノイズが入ったりなど、汚くなってしまいます。その代わりにファイルサイズを小さく圧縮することができます。可逆圧縮では、画像は元の綺麗なままですが、ファイルサ

イズは非可逆圧縮のものと比較して大きくなってしまいます。昨今はハードディスクの容量も大きくなり、ファイルサイズが問題になることは基本的に無いので、可逆圧縮である PNG 形式を使います。何か特別に事情があって容量を小さくしなければならない時には JPEG 形式を使うのかもしれません。 PNG 形式にはインデックスカラーが使える、という特徴もあります。吉里吉里で「領域画像」というものを使う際にはインデックスカラーである必要があります。この本では領域画像を使わないので説明は割愛します。

TLG5 と TLG6 は吉里吉里独自の画像形式で、ほとんどの場合これを使います。どちらも可逆圧縮なので綺麗なまま画像を使えます。 TLG5 は PNG 形式よりもサイズが大きくなってしまいますが、画像の表示が少しだけ速くなります。 TLG6 は PNG 形式よりもサイズが小さくなり、 TLG5 よりは遅いですが PNG よりは画像の表示も速くなります。どちらかはケースバイケースになるかもしれませんが、私はたいてい TLG6 を使っています。

TLG5 と TLG6 は、吉里吉里独自の形式のためフォトショップや SAI など一般の画像編集ソフトでは使えません。まずは PNG 形式で保存しておき、吉里吉里付属のツールを使って変換することになります。

■背景の表示

まずは背景画像を表示してみます。背景のサンプルとして使うのは、サンプル素材の「 bgimage 」フォルダにある「 bg 街 _ 昼 .png 」などの画像です。フォルダの画像を全て開発用フォルダ内の「 bgimage 」フォルダに全てコピーしてください。

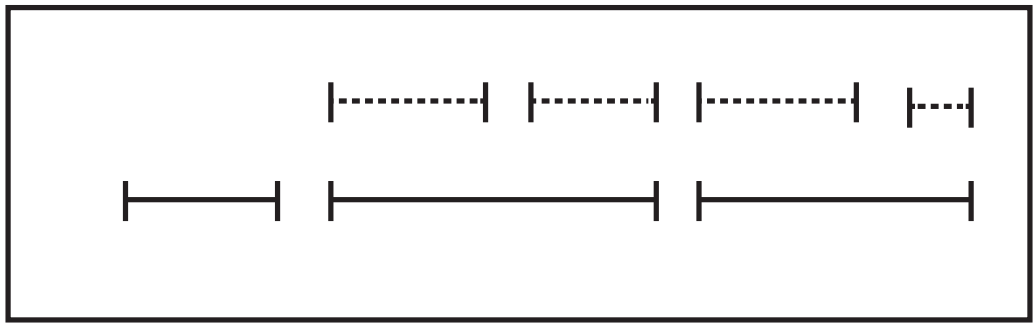
背景画像のファイル名は必ず「 bg[場所名]_[時間名] 」という形になります。サンプルでは場所名が「街」「向日葵」「リビング」、時間名が「昼」「夕」「夜」「共通」となっています。

背景を表示するには、スクリプトを書く前に「場所名」と「時間名」をそれぞれ定義しておく必要があります。定義には本来であれば TJS スクリプトを書く必要がありますが、簡単にするためにツールを作ってあります。「開発用フォルダ」の「 tools 」フォルダの「 autosetting.bat 」を実行してください。「プロジェクトフォルダの選択」というダイアログが出てきますので、「開発用フォルダ」の「 data 」フォルダを選択して「 OK 」をクリックします。「 bgimage 」フォルダの中の画像を見て、必要な場所と時間をそれぞれ定義してくれます。

定義ができたらスクリプトを書いて実際に表示してみます。

```
*start| スタート
@stage stage= 街 stime= 昼
場所を街、時間を昼に変更しました。

@stage stage= 街 stime= 夜
場所を街、時間を夜に変更しました。

@stage stime= 夕
時間を夕に変更しました。

@stage stage= 向日葵
場所を向日葵に変更しました。
```

@ （半角アットマーク）から始まるのはコマンド行です。コマンド行はタグとも言います。画像を表示したり、音を鳴らしたりなど、何かをするには必ずタグを使います。



タグは、先頭の「タグ名」と、それに続く「属性」から成ります。 @stage stage= 街 stime= 昼は、「 stage 」がタグ名、「 stage= 中庭」と「 stime= 昼」が属性です。タグ名とそれぞれの属性は半角スペースで区切られます。さらに、属性については属性名と属性値というものにわけられます。 =(半角イコール) の前の部分が属性名、後の部分が属性値です。

背景を表示するには、「 stage 」という名前のタグを使います。「 stage 」属性に場所、「 stime 」属性に時間を属性値として指定します。そうすると、指定した場所と時間にゲームの状態が変更され、背景の画像もそれに合わせて変わります。 @stage stage= 街 stime= 昼というタグでは、場所が街、時間が昼に変更されるので、背景には「 bg 街 _ 昼 .png 」が表示されます。

また、必要のない属性は省略することもます。 @stage stime= 夕は、 stage 属性を省略しているので、場所は街のままで時間だけ夕に変更されます。@stage stage= 向日葵は、時間は夕のままで場所だけ向日葵に変更しています。

```
*start| スタート
@stage stage= 白 stime= 昼
場所を白、時間を昼に変更しました。

@stage stime= 夕
時間を夕に変更しました。
背景白 _ 共通のままです。
```

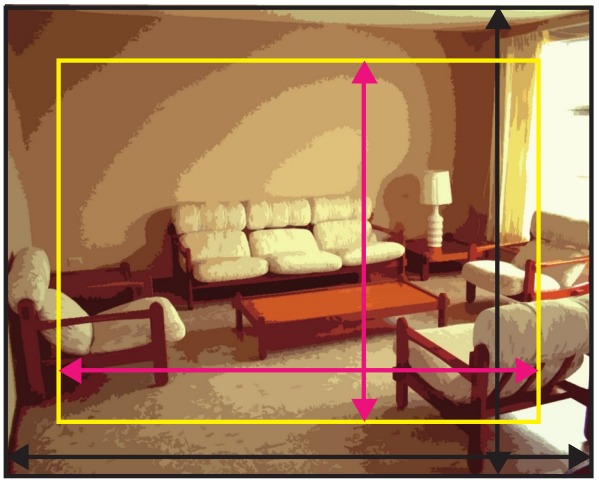
「白 _ 共通 .png 」と「黒 _ 共通 .png 」の時間は「共通」となっています。「共通」については、その名の通り、どの時間の場合でも共通に同じ画像が表示されます。上のスクリプトでも時間に関係なく背景は「白 _ 共通」が表示されます。

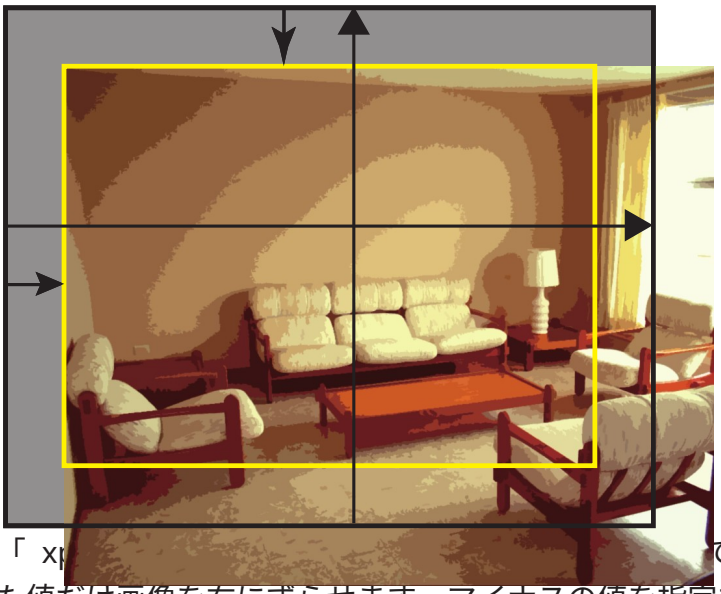
■画像の位置

背景画像は画面中央に表示されます。サンプルの「 bg リビング _ 昼 .png 」が画面サイズより大きいので、それで試してみます。

```
*start| スタート
@stage stage= リビング stime= 昼
リビング、昼の背景を表示。
画像の中央が表示されています。

@stage xpos=100 ypos=-100
画像の位置を変更しました。
左上部分が表示されています。
```

デフォルトのウィンドウサイズは横 800 ピクセル × 縦 600 ピクセル、リビングの画像のサイズは横 1000 ピクセル × 縦 800 ピクセルです。正確にいうと背景画像は、画像の中央がウィンドウの中央になるように表示されます。なので、最初の @stage stage= リビング stime=昼 で下の画像の「ウィンドウ枠」の内部が表示されます。





画像の位置は、「 xp できます。 xpos が横方向で、指定した値だけ画像を右にずらせます。マイナスの値を指定すると左にずらすこともできます。 ypos は縦方向で、指定した値だけ上にずらせます。マイナスの値を指定すると下にずらせます。 @stage xpos=100 ypos=-100 では、右に 100px 、下に 100px ずらすことで画像の左上の部分を表示しています。

■ピクセル (pixel)

吉里吉里 /KAGEX での画像位置の指定は、基本的にピクセル (px) が単位になります。ピクセルはモニタで色を表示する際の最小単位です。例えば、ディスプレイの画面解像度 1920 × 1080 というのは横に 1920px 、縦に 1080px 表示するということです。モニタに限らず、デジタル画像のサイズはピクセルが単位になっています。デジカメの何万画素とかいうのもピクセルのことです。 1000px × 1000px サイズの写真が撮れるなら 1,000,000 画素になります。

ピクセルは論理的な単位なので、物理的に何センチかというのは決まっていません。同じ 100px × 100px の画像でも、モニタによって表示される大きさが異なってきます。ピクセル数と実際の物理的なサイズは dpi(dots per inch) という単位で換算されます。ドット (dot) とピクセルはおなじようなものです。印刷やスキャンをするときによく使われる単位です。 10 インチ × 10 インチの絵を 300dpi(1 インチに対して 300 ドット) でスキャンすると 3000px × 3000px のデジタル画像になります。

吉里吉里 /KAGEX の解像度 (吉里吉里のウィンドウ内のピクセル数) は前述のとおり 800 × 600 になっています。ウィンドウの枠をひっぱるとウィンドウサイズが変わってウィンドウ内の実際のピクセル数も変わったりするのですが、 KAGEX スクリプトを使っているうちは自動的に何とかしてくれているので意識せずに済みます。常に拡縮されていない状態で考えればいいです。 TJS でカスタマイズするようになると

少し気をつける必要が出てきます。

商業ゲームだと画面サイズもどんどん大きくなっています。極端なものだと 1900 × 1200 なんてのもあったりします。同人ではまだ 800 × 600 が主流な気がします。画面サイズが大きくなると嬉しいのは、フルスクリーンでプレイしても絵が綺麗なことですね。 800 × 600 をフルスクリーンにすると、モニタによっては絵が汚くなってしまうかもしれません。製作側としては画像を大きく作る必要が出てくるので大変です。ウェブのフリー素材を使う場合は 800 × 600 のサイズが多かったりするのでその点もネックになるのかもしれません。

もう 1 つトピックとしてアスペクト比があります。アスペクト比とは解像度の横と縦の比率のことです。 800 × 600 ならアスペクト比 4:3 、流行りのワイド対応なら 1280 × 720 でアスペクト比 16:9 、などとなります。ワイドでは画面が横長で文字が読みづらい、絵がおかしくなるなど今までの感覚で作ると問題が出てきます。ワイド化のせいで CG が斜めになりすぎる、というネタもあったりします。 (参考： 2ch 「エロゲーは 4:3 で良い。ワイド化は誤り」スレ http://pele.bbspink.com/test/read.cgi/erog/1299685511/) 参考 URL では主観視点云々という話とつながってますが、映画的に三人称視点で鑑賞している身としてもカメラが傾いているのはおかしいです。ワイド化が誤りとは言いませんがもうちょっとやりようはあるのかなーと。テキスト表示は下端から横端に移して画面を縦に使う、というのが 1 つ。スクリプトを組んでいると、立ち絵の絶対領域がテキスト枠に隠れているのがもったいない、とおもいます。何ならワイド化やめて縦長画面にしてしまってもいいのではないでしょうか。縦長画面に表示されてる女の子って、それだけでかわいさ 3 割増なのです。 iPad はデバイスの画面自体が縦長でその点期待です。パソコンでも縦 1000 あれば縦長でゲーム作れるはずです。誰かやりましょう。ノートパソコンの方はごめんなさい。

関係ないですが主人公＝自分だと思ってプレーしてる方って結構いるんですねえ。自分にはその発想自体がなかったです。主人公と女の子がキャッキャウフフしているのを見ているだけ、という感覚に近い。演出付ける際も、主人公視点という意識は薄く、どちらかというと三人称視点で動かしてる気がします。主人公の立ち絵もあればいいのに、と思うこともしばしば。

閑話休題。スクリプト解説に戻ります。

■立ち絵の表示

背景の次は立ち絵画像を表示してみます。サンプル素材の「 fgimage 」フォルダの「 fg しおり _ ポーズ a_ 私服 _ 笑顔 _0.png 」などの画像を使います。全て開発用フルダの「 fgimage 」フォルダにコピーしてください。

立ち絵画像のファイル名は「 fg[キャラ名]_[ポーズ名]_[服装名]_[表情名]_[表示レベル] 」という形になります。サンプルでは、キャラ名は「しおり」と「かえで」、ポーズ名は「ポーズ a 」と「ポーズ b 」……などとなっています。ファイル名に英字を使う場合は必ず小文字にしてください。「ポーズ A 」ではなく「ポーズ a 」である必要があります。

立ち絵については、「キャラ名」、「ポーズ名」、「服装名」、「表情名」、「表示レベル」についてそれぞれ定義しておく必要があります。こちらも autosetting.bat で自動的に定義できます。立ち絵画像をコピーしてから、 autosetting.bat を実行してください。

いま使うサンプル画像は 1 枚の画像で立ち絵が表示できます。「表情合成型立ち絵」といいます。この場合、ポーズ名、服装名、表情名の区分はあまり重要ではありません。自前の立ち絵画像を使う場合にも、特に気をつける点もなくポーズが異なっていたらポーズ名、服装が異なっていたら服装名、表情が異なっていたら表情名の部分を適当に変えておけば OK です。キャラ名、の部分は説明不要かと思います。キャラクタごとの名前になります。表示レベルは立ち絵のサイズで分類することになると思います。表示レベルは必ず 0 以上の半角数字で、数字が大きいほど立ち絵の中でも前の方に表示されます。実際のスクリプトはこれから見ていきます。

立ち絵を「土台部分の画像＋表情部分の画像」と 2 枚の画像を組み合わせて表示する形式にすることもできます。こちらは「表情分離型立ち絵」といいます。こうすると画像の容量が小さくなったり素材管理が楽になったりします。こちらの場合の画像の作り方は後述します。 1 度画像を作って定義してしまえば、合成型も分離型もスクリプトでの使い方は全く同じになります。

```
*start| スタート
@stage stage= 街 stime= 昼
背景を表示しました。

@char name= しおり pose= ポーズ a dress= 制服 face= 笑顔 level=0
【しおり】「立ち絵を表示しました。」

@char name= しおり pos= 消
立ち絵を消去しました。

@char name= しおり face= 驚く
表情を変更してもう一度表示。
```

立ち絵を表示するには「 char 」タグを使います。「 name 」属性にキャラ名、「 pose

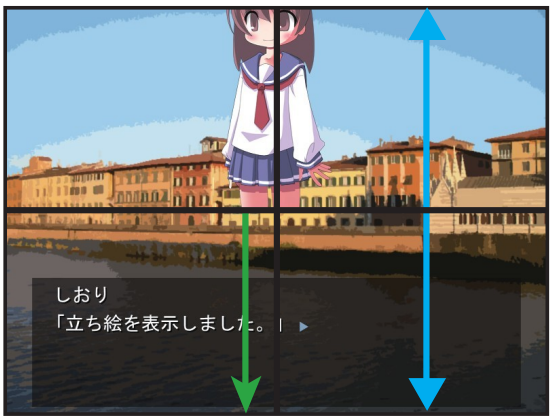
属性にポーズ名、「 dress 」属性に服装名、「 face 」属性に表情名、「 level 」属性に表レベルを指定します。属性の数が多いですが、「 name 」属性以外は最初にまとめて指定しておくだけで、後は変更する属性のみ指定すれば十分です。「 name 」属性はどの立ち絵を操作するのか判定するため省略できません。

「 pos 」属性では立ち絵の表示状態を変更できます。「 pos= 消」とすると立ち絵が非表示になります。表示する際は表情やポーズなどを変更すれば自動的に表示されます。

背景での表示、非表示の方法を説明していませんが、背景では「黒 _ 共通 .png 」のような黒ベタの画像を表示させればいいので背景自体を非表示にすることはしません。

■立ち絵 x オフセット定義

先ほどのスクリプトを実行してみると、左下の画像のように立ち絵が変なところに表示されます。実は立ち絵は、立ち絵画像の下端中央が画面中央になるように表示されます。何故この位置になるのか詳しい話は 4 章で説明します。とりあえず今回の場合は、右下の画像のように下に 300px ずらすとちょうど良くなります。



ずらすには、背景と同じく xpos 属性と ypos 属性を使ってもいいのですが、全ての立ち絵の位置を一括でずらすように設定することもできます。立ち絵の最初のズレを直すにはそちらの機能を使った方が便利です。

プロジェクトフォルダの「 init 」フォルダの「 envinit.otjs 」をテキストエディタで開いてください。これは、何度か使用した autosetting.bat の動作を設定するファイルになります。開いたら、次のように書かれた部分を探してください。

```
// ■ yoffset
//  表示レベルごとに、立ち絵の Y 方向オフセット ( ずらし量 ) を指定します。
yoffset[0] = 0;
yoffset[1] = 0;
yoffset[2] = 0;
```

オフセットは、表示レベルごとに設定できます。 yoffset[表示レベル]= オフセットの形式で設定していきます。初期設定では表示レベル 0 、 1 、 2 でオフセット 0(全くずらさない) となっています。サンプルで使っているのは表示レベル 0 と 1 なので、そのオフセットを下方向 300px に設定すれば OK です。 yoffset では、 ypos 属性とは逆に正の値を指定すると下、負の値を指定すると上にずれるようになります。

```
// ■ yoffset
//  表示レベルごとに、立ち絵の Y 方向オフセット ( ずらし量 ) を指定します。
yoffset[0] = 300;
yoffset[1] = 300;
yoffset[2] = 0;
```

このように変更すれば OK です。変更したら、 tools フォルダの autosetting.bat をもう一度実行すると変更が適用されます。ゲームを起動して立ち絵の位置がずらされていることを確認してください。

■表示レベル

```
*start| スタート
@char name= しおり pose= ポーズ a dress= 制服 face= 笑顔 level=0
【しおり】「立ち絵を表示しました。」

@char name= かえで pose= ポーズ a dress= 私服 face= 無表情 level=1
【かえで】「もう 1 人の立ち絵を表示しました。」

@char name= かえで level=0 xpos=80
【かえで】「表示レベルと表示位置を変更しました。」

@char name= しおり front
【しおり】「前に移動します。」

@char name= しおり back
【しおり】「後ろに移動します。」
```

立ち絵は、表示レベルが大きいほど前に表示されます。かえでの立ち絵はlevel=1で、しおりの level=0よりも大きいのでしおりの立ち絵の前に表示されます。その後 @char

name= かえで level=0 xpos=80 表示レベル 0 に変更しても、後から変更されたものが前になるため、かえでの立ち絵が前に表示されたままです。

表示レベルが同じ立ち絵の前後関係は、「 front 」属性と「 back 」属性で変更できます。@char name= しおり front=1 のように属性値は省略して属性名のみで使います。 front 属性で同じ表示レベルの中で一番前、 back 属性で一番後ろに表示順を変更できます。

■イベント絵の表示

次にイベント絵を表示します。サンプル素材の「 evimage 」フォルダの「 ev 発見 .png 」を使います。イベント絵のファイル名は「 ev[イベント名] 」という形式になります。

イベント名についても autosetting.bat で定義しておく必要があります。サンプル画像をプロジェクトフォルダの「 evimage 」フォルダにコピーしたら、 autosetting.bat を実行しておいてください。

```
*start| スタート
@stage stage= 街 stime= 夕
@char name= しおり pose= ポーズ a dress= 制服 face= 笑顔 level=0
背景と立ち絵を表示しました。

@event file= 発見
イベント絵を表示しました。

@stage stage= 向日葵
背景を変更しました。
```

イベント絵を表示するには「 event 」タグを使います。「 file 」属性にはイベント名を指定します。イベント絵は表示レベル 4 と表示レベル 5 の立ち絵の間に表示されるので、 @event file= 発見 でイベント絵を表示すると背景と立ち絵は後ろに隠れて見えなくなります。 @stage stage= 向日葵 のように stage タグで背景または時間を変更するとイベント絵は非表示になります。

イベント絵の表示位置は、背景と同じく画像の中央がウィンドウの中央となります。位置をずらすには xpos 属性と ypos 属性を使います。

■その他の画像の表示

背景、立ち絵、イベント絵以外の画像を表示したい場合があります。サンプル素材の「 image 」フォルダの「星 .png 」を表示してみます。プロジェクトフォルダの「 image 」フォルダにコピーしておいてください。

その他の画像のファイル名に決まりはないので、自分でわかりやすいファイル名であれば構いません。事前に autosetting.bat などで定義しておく必要もありません。

```
*start| スタート
@layer name= お星様 file= 星 show
星を表示しました。

@layer name= お星様 hide
星を非表示にしました。

@layer name= お星様 show
もう一度表示しました。
```

その他の画像は「環境レイヤ」に表示します。環境レイヤは「 layer 」タグを使うことで表示できます。「 name 」属性に環境レイヤの名前、「 file 」属性に画像ファイル名を指定すると、指定した名前の環境レイヤに画像ファイルを読み込まれます。名前は何でもいいので適当に付けてください。名前が違うと別の環境レイヤになるので、同じ画像を複数枚表示させることもできます。環境レイヤは、属性値を省略した「 show 」属性で表示、「 hide 」属性で非表示にできます。

```
*start| スタート
@layer name= お星様１ file= 星 show
星を表示しました。

@layer name= お星様２ file= 星 show xpos=20 level=1
星をもう一枚表示しました。

@layer name= お星様１ level=2
星１を前にしました。
```

環境レイヤにも、立ち絵と同じく表示レベルがあります。「 char 」タグではなく「 layer 」タグになっているだけで全く同じです。「 front 」と「 back 」属性も同じように使用できます。

背景、イベント絵と同じく環境レイヤも画像の中央がウィンドウの中央となる位置に表示されます。「 xpos 」と「 ypos 」属性で位置をずらせるのも同じです。

実は「 file 」属性でファイルを読み込むと自動的に表示されるので最初の @layer name= お星様1 file= 星 show の「 show 」属性は不要だったりします。わかりやすいように「 show 」を付けるようにしていますが面倒なので付けなくても構いません。逆に「 file 」

で読み込むだけで表示したくない場合は「 hide 」属性を付ける必要があります。

■コメント

```
*start| スタート
; コメントのサンプルです
; 先頭が ; の行は無視されます
@stage stage= 街 stime= 昼
背景を表示しました。
```

演出や演技の指示など、スクリプト中にメモ書きをしたい場合があります。「 ; 」(半角セミコロン) を先頭につけると、その行は「コメント」としてゲームの動作に影響を与えなくなります。セミコロンがついた行は単純に無視されるので何でも書けます。

ついでに言うと、 TJS スクリプトでは「 // 」(半角スラッシュ 2 つ) がついた行がコメントになります。 envinit.otjs では、 表示レベルごとに、立ち絵の Y 方向オフセット (ずらし量) を指定します。のような行がありましたが、コメントによるメモ書きになっています。

■省略記法

背景、立ち絵、イベント絵、その他とゲーム中で使用する画像の出し方を紹介しました。これらは非常に頻繁に使いますが、タグ名や属性が長くて大変なので省略して使えます。

```
*start| スタート
; @stage stage= 街 stime= 昼 と同じ
@ 街 昼
背景を表示しました。

; @stage stage= 向日葵 と同じ
@ 向日葵
場所を変更しました。

; @stage stime= 夕 と同じ
@ 夕
時間を変更しました。
```

背景については、「 stage 」というタグ名と、「 stage 」、「 stime 」の属性名を省略場所名または時間名をタグ名とします。 xpos のような他の属性名は省略できないので @stage xpos=200 または @ 街 xpos=200 のようになります。

```
*start| スタート
; @char name= しおり pose= ポーズ a dress= 制服 face= 笑顔 level=0
@ しおり ポーズ a 制服 笑顔 奥
【しおり】「立ち絵を表示しました。」

; @char name= しおり pose= ポーズ a dress= 私服 face= 無表情 level=1
@ かえで ポーズ a 私服 無表情 前
【かえで】「もう一人立ち絵を表示しました。」

; @char name= かえで face= 笑顔
@ かえで 笑顔
【かえで】「表情を変更しました。」

; @char name= かえで pos =消
@ かえで 消
立ち絵を消去しました。
```

立ち絵については、「 char 」というタグ名と、「 name 」、「 pose 」、「 dress 」、「 level 」、「 pos 」の属性名を全て省略できます。「 name 」属性に指定していたキャラクタ名をタグ名として使えます。

表示レベルを指定する「 level 」属性を省略するには、表示レベルの名前を定義しておく必要があります。 init フォルダの envinit.otjs を開いて、以下のようなところを探してください。

```
// ■ levels
// 表示レベルごとに、立ち絵の表示レベルに名前をつけます。
levels[0] = " 奥 ";
levels[1] = " 前 ";
levels[2] = " 手前 ";
```

levels[表示レベル] = " 表示レベルの名前 " という形で定義します。表示レベルの名前は ""(半角ダブルクオート) で囲みます。デフォルトでは「奥」が level=0 「前」が level=1 「手前」が level=2の省略として使えます。必要なら書き換えて autosetting.bat を実行すると、定義した名前がスクリプト中で使えるようになります。本書のサンプルは変更せずこのままで使います。

立ち絵については、「 xpos 」属性にも名前を定義して省略することができます。こちらも「 envinit.otjs 」を見てください。

```
// ■ xpos
// X 方向の位置に名前をつけます。
xpos. 左外  = -400;
xpos. 左奥  = -300;
xpos. 左    = -200;
xpos. 左中  = -100;
xpos. 中    =  0;
xpos. 右中  =  100;
xpos. 右    =  200;
xpos. 右奥  =  300;
xpos. 右外  =  400;
```

xpos. 名前 = xpos の値形で定義します。デフォルトでは「左外」が xpos=-400 「左奥」が xpos=-300 「左」がxpos=-200……の省略となっています。こちらも必要なら書き換えて、「 autosetting.bat 」を実行しておいてください。本書のサンプルではこのままで使います。

```
*start| スタート
@ しおり ポーズ a 制服 笑顔 奥 左
【しおり】「立ち絵を表示しました。」

@ かえで ポーズ a 私服 無表情 奥 右
【かえで】「もう一人立ち絵を表示しました。」

@ しおり 消
@ かえで 消
立ち絵を消去しました。
```

実際の使い方は上のようになります。服装や表情と同じく並べて指定するだけです。

```
*start| スタート
; @event file= 発見 と同じ
@ 発見
イベント絵を表示しました。
```

イベント絵については、「 event 」というタグ名と「 file 」属性を省略できます。イベント名がタグ名になります。位置を変更する際は @ 発見 xpos=300 のようになります。イベント絵であることが分かりやすいように @ev 発見 タグ名に「 ev 」を付けることもできます。使いやすい方を使ってください。

```
*start| スタート
@layer name= お星様 file= 星 show
星を表示しました。

; @layer name= お星様 hide と同じ
@ お星様 hide
星を非表示にしました。

; @layer name= お星様 show と同じ
@ お星様 show
もう一度表示しました。
```

環境レイヤについては、タグ名の「 layer 」と、「 name 」属性を省略できます。「 nam
に指定していた環境レイヤ名がタグ名になります。

ただし、初めて使う環境レイヤ名の場合は省略することができません。初めて使う
う際に指定した名前の環境レイヤが作られるので、既に作られている場合にのみ省略
することができます。

```
*start| スタート
; ここで「お星様」という名前の環境レイヤが作られます
; @ お星様 file= 星 show と省略はできません
@layer name= お星様 file= 星 show
星を表示しました。

; @layer name= お星様 xpos=100 と同じ
@ お星様 xpos=100
星の位置を変更しました。

; @layer name= お星様 delete と同じ
@ お星様 delete
環境レイヤ自体を削除しました。

; 次に使うときは再び「 @layer 」で環境レイヤを作る必要があります
@layer name= お星様 file= 星 show
星を表示しました。
```

「 hide 」属性で非表示にするのではなく、@ お星様 delete と「 delete 」属性を使うこと
で作成された環境レイヤを削除することもできます。属性値は省略します。使わなく
なった環境レイヤが残るのが気になる場合は削除してもいいかもしれません。削除し

た場合は、次に使う際に再び省略せずに「 @layer 」を使って環境レイヤを作成させる必要があります。

■BGM の再生

画像はひと通りやったので、 BGM を再生してみます。

サンプルの「 bgm 」フォルダの「 bgm 湖畔の村 .ogg 」と「 bgm 冬の息吹 .ogg をプロジェクトフォルダの「 bgm 」フォルダにコピーしてください。 bgm は、「 bgm[bgm 名] 」という形のファイル名になります。 BGM については autosetting.bat などを実行する必要はありません。

```
*start| スタート
@bgm play=bgm 湖畔の村
BGM を再生しています。

@bgm play=bgm 冬の息吹 loop=false
BGM を切り替えました。ループ再生はしません。

@bgm stop
BGM を停止しました。
```

BGM を再生するには「 bgm 」タグを使用します。「 play 」属性に BGM ファイル名を指定すると、そのファイルが BGM として再生されます。 BGM が最後まで再生し終わると自動的に最初からループ再生されます。「 loop 」属性に「 false 」を指定するとループ再生しなくなります。属性値を省略した「 stop 」属性で BGM の再生を停止できます。

```
*start| スタート
; @bgm play= 湖畔の村 と同じ
@bgm 湖畔の村
BGM を再生しています。

; @bgm play= 冬の息吹 loop=false と同じ
@bgm 冬の息吹 loop=false
BGM を切り替えました。ループ再生はしません。

@bgm stop
BGM を停止しました。
```

BGM については再生する際にタグ名の bgm と play 属性を省略できます。@bgm 湖畔の村のように BGM ファイル名をタグ名とするとその BGM が再生されます。

■音楽ファイルの形式について

吉里吉里で主に再生できる音楽ファイルの形式は、 PCM WAV(拡張子は wav) と OggVorbis(拡張子は ogg) になります。

wav は音楽 CD でも使われている形式で、基本的に無圧縮のため音質は良いですがファイルサイズが大きくなってしまいます。一般的な設定の wav で 1 分 10M ほどになります。ファイルサイズが大きくなってもいい場合は使うこともあります。

ogg は非可逆圧縮のため音質は悪くなりますがファイルサイズを小さくすることができます。ゲーム中では大抵の場合こちらを使います。一般に使用されている mp3 とは異なりライセンス料がかかりません。 wav や mp3 など、他の形式から ogg に変換するツールは検索すると色々でてくるので好きなものを使ってください。

ogg や mp3 は非可逆圧縮のため、ファイルを編集するごとに音質がどんどん劣化していきます。エフェクトをかけたり音量調整をしたりなどは必ず元の wav ファイルでやるようにしてください。編集が終わってから、最後に ogg に変換してゲーム中で使用します。

実は 5.1ch サラウンドにも対応しているようです。誰かやってみたら楽しいかも。

wav と ogg の他にも MIDI 、 CD-DA 、独自形式の TCWF などが使えるらしいです。使わないので解説しません。

■効果音の再生

次に効果音 (SE = Sound Effect) を再生してみます。サンプルの「 sound 」フォルダの「 se ボタン .ogg 」と「 se 雨 .ogg 」をプロジェクトフォルダの「 sound 」フォルにコピーしてください。効果音は「 se[効果音名] 」という形のファイル名になります。効果音についても autosetting.bat などを実行する必要はありません。

```
*start| スタート
@se play=se 雨 loop=true
SE をループ再生しています。

@se play=se ボタン
もう 1 つの SE をワンショット再生しました。

@se name=se 雨 stop
SE を停止しました。
```

効果音を再生するには「 se 」タグを使用します。「 play 」属性に効果音ファイル名を指定すると、そのファイルが効果音として再生されます。効果音は BGM と異なりそのままではループ再生されません。ループせずに 1 回だけ鳴らすことをワンショット再生と言ったりもします。「 loop 」属性に「 true 」を指定すると効果音でもループ再生させることができます。

BGM と効果音での大きな違いは、 BGM が同時に 1 曲しか再生できないのに対し、効果音は同時にいくつも再生することができます。上のスクリプトでは「 se 雨」をループ再生している間に「 se ボタン」も再生しています。同時に再生できる効果音の数はデフォルトでは 3 つになっています。設定で増やせますが、 4 つ以上同時再生したいケースは稀です。

再生を停止する場合は「 stop 」属性を使用しますが、同時にいくつも再生できる効果音の場合は「 name 」属性でどの効果音を停止するかを指定する必要があります。@se name=se 雨 stopのようになります。

```
*start| スタート
; @se play=se 雨 loop=true と同じ
@se 雨 loop=true
SE をループ再生しています。

; @se play=se ボタン と同じ
@se ボタン
SE をもう一つ再生しました。

; @se name=se 雨 stop と同じ
@se 雨 stop
SE を停止しました。
```

se についてはタグ名の se と play 属性を省略できます。効果音ファイル名をタグ名とするとそのファイルが効果音として再生されます。停止する場合にも、 name 属性を省略し、ファイル名をタグ名として stop 属性をつけると停止できます。

■true と false

BGM と SE に共通の属性である loop 属性には「 true 」または「 false 」を指定し日本語では true = 真、 false = 偽などと訳されます。 true または false を指定する属は他にもたくさんあります。 true を指定するとその属性の機能がオンに、 false を指定するとオフになる、と思っておくといいです。「 loop 」属性では true を指定するとループ再生がオンに、 false を指定するとループ再生がオフになります。先ほどのス

クリプトでは BGM では loop=false、 SE では loop=true しか使っていませんが、 BGM で loop=true、 SE で loop=false と指定することもできます。ただし、前述のとおり BGM ではデフォルトでループ再生がオン、 SE ではオフになっているのでわざわざ指定する意味はありません。

■属性値 true の省略

属性は = で区切られた属性名と属性値から成りますが、属性値が true の場合は属性値を省略することができます。 @se 雨 loop は @se 雨 loop=true と同じです。立ち絵の表示順を変更する際に front 属性や back 属性の属性値を省略したのを覚えているでしょうか。実はこれらも front=true や back=true を省略したものです。前に持ってくる機能をオンにする、後ろに持ってくる機能をオンにする、といった感じの意味になります。true を指定できるということは front=false または back=false と指定することもできるのですが、この場合は何も起きないので全くの無駄になります。結局 front または back と属性値を省略して使うことになります。 layer タグでの show 属性と hide 属性についても全く同じことです。

■スクリプトの実行順

KAGEX スクリプトはほとんどタグからできています。ここまで stage 、 char 、 event 、……などのタグを紹介してきました。テキストはそのまま書くだけで表示されていますが、実はこれもタグだったりします。テキストの表示には「 ch 」というタグが使われています。テキストは大量に使うのに、いちいちタグを書いていては面倒なので省略してそのまま書けるようになっています。改行やページ送りなども、それぞれタグが使われています。

KAGEX スクリプトは上から順番に実行されます。当たり前ですが重要です。

```
*start| スタート
@stage stage= 街 stime= 昼
背景を表示しました。

@char name= しおり pose= ポーズ a dress= 制服 face= 笑顔 level=0
立ち絵を表示しました。
```

まず背景を表示する @stage stage= 街 stime= 昼 が実行されます。表示し終わったら 背景を表示しました。という文字を表示する「 ch 」タグが実行されます。次に空行が挟まっているので、「プレイヤーがクリックするのを待つタグ」が自動的に実行され、スクリプトの実行が一時停止します。

ここでスクリプトの実行が一時停止されている、というのも知っておくといいです。

何かしらのタグで停止または一時停止されない限り、 KAGEX スクリプトは上からどんどん実行されていきます。

プレイヤーがクリックするとスクリプトの実行が再開され、「ページ送りをするタグ」が自動的に実行されてテキストが消去されます。その後@char name= しおり pose= ポーズ a dress= 制服 face= 笑顔 level=立ち絵が表示され、最後に「 ch 」タグで立ち絵を表示しました。と表示されます。最後まで実行が終わったのでスクリプトの実行は停止します。

選択肢でシナリオが分岐したりなど、上から順番に実行しているだけでは済まない場合があります。選択肢 A と B が選ばれた場合でそれぞれスクリプトを実行する位置を変えなければなりません。

```
*start| スタート
スタートです。

@next target=* ラベル 1

* ラベル 2
ラベル 2 です。

@next target=*start

* ラベル 1
ラベル 1 です。

@next target=* ラベル 2
```

まずは選択肢などを使わずに、単純にスクリプトの実行位置を変えてみます。それには「 next 」タグを使用します。「 target 」属性にラベル名を指定すると、次はそのラベルからスクリプトが実行されます。ラベルとは *startや * ラベル2のように *(半角アスタリスク) から始まる行です。ラベルは next タグに限らず、スクリプトの実行位置を変更するための目印の役割を果たします。もし「 01.ks 」の 100 行目、などと行数で指定しようとすると、タグを付け足して 1 行ずれるだけで実行したい位置もずれていってしまって大変です。ラベルを使うことで行数がずれても大丈夫なようになっています。

今まで説明してこなかった *start| スタート もラベルです。後ろに |(半角縦線) がついていますが、これは無視してください。また後回しになってしまいますがすぐに説明します。ラベル名は | の前の「 start 」になります。吉里吉里が起動されると、 KAGEX

の準備やら何やらが終わってから、「 01.ks 」の*startラベルから KAGEX スクリプトの実行が始まる、という仕組みになっています。

上のスクリプトでは、 スタートです。が表示された後、 @next target=* ラベル1によって、 * ラベル1へとスクリプトの実行位置が変更されます。ラベル 1 へ「ジャンプ」する、とも言います。よって、 ラベル 1 ですが表示され、同じく * ラベル2へとジャンプし、 ラベル 2 ですが表示されます。最後に @next target=*startで再び *startから実行されるので「スタートです。」「ラベル 1 です。」「ラベル 2 です。」……がクリックするたびに延々と表示され続けることになります。

ラベル名には「 start 」や「ラベル 1 」のように好きな名前をつけれますが、スペースは使えません。

```
*start| スタート
スタートです。

@next target=* ラベル 1 storage=02.ks
```

「 storage 」属性を使うと、別のファイルのスクリプトを実行することができます。上のスクリプトは今までどおり「 01.ks 」に書いてください。

プロジェクトフォルダの「 scenario 」フォルダに新しいテキストファイルを作成して、拡張子も含めて「 02.ks 」に名前を変更します。変更したら以下のスクリプトを「 02.ks 」に書いてください。

```
* ラベル 1
02.ks のスクリプトが実行されています。

@next target=*start storage=01.ks
```

ゲームを起動すると、いつもどおり「 01.ks 」の*startから実行が始まります。 スタートです。が表示された後、 @next target=* ラベル storage=02.ksによって、「 02.ks 」の* ラベル 1へとジャンプします。そして @next target=*start storage=01.ksによって「 01.ks 」の*startへと戻ってきます。

ファイル名は 01.ks や 02.ks など「番号 +.ks 」としていますが、番号にしなければならないという決まりはありません。「しおり編 .ks 」や「エピローグ .ks 」などわかりやすい名前をつけることができます。 @next storage= しおり編 .ks target=* ?のようにstorage 属性に指定すればそのファイルのスクリプトを実行できます。

■セーブデータ名の指定

```
*start| スタート
セーブデータ名は『スタート』です。
メニューからセーブしてみてください。

* ラベル 0| ０００１
セーブデータ名は『０００１』です。

* ラベル１
ここでもセーブデータ名は『０００１』です。

*| ０００２
ここのセーブデータ名は『０００２』です。
```

ラベルの「 | 」(半角縦棒) の後ろはセーブデータの名前になります。ゲームを起動して、メニューの「データ」の「セーブ」からセーブしてみてください。どの時点でセーブするかによって「ロード」から見れるデータ名が変わっていることがわかると思います。

* ラベル 0| ００を通過するとセーブしたときのデータ名が「０００１」に変わります。次の * ラベル では セーブデータ名がついていません。この場合は変更されず「０００１」のままです。 *| ０００２ では、ラベル名が省略されています。この場合は target 属性に指定するラベル名がわからないため、 next タグの実行位置の目印としては使えません。セーブデータ名のみを変えたい場合にはこのような省略ができます。

```
*start| スタート
セーブデータ名は『スタート』です。

ラベルがついていない次のテキストです。

その次のテキストです。
```

next タグの際に説明したように、スクリプトの実行位置を変更するにはラベルを目印とする必要があります。これはセーブしたデータをロードするときにも同じです。なので、 ラベルがついていない次のテキストです。 でセーブしたデータをロードする場合、直接そこからスクリプトを実行させることができません。 KAGEX では、*start| スタート にジャンプしておいて、そこから実際にセーブした ラベルがついていない次のテキストです。 までスクリプトの実行を早送りする、という方法でロードします。

2 、 3 行程度の早送りであれば一瞬で終わりますが、 1000 行、 2000 行と早送りを

するには時間がかかってしまいます。このため、前のラベルがあまりに遠いところでセーブしたデータをロードすると、早送りに時間がかかって少し止まったような動作になってしまいます。

```
*start| スタート
セーブデータ名は『スタート』です。

*|
ラベルがついていない次のテキストです。

*|
その次のテキストです。
```

これを防ぐため、他に必要が無くてもところどころにラベルを挟むようにします。私の場合は 300 ～ 500 行に 1 つはラベルを置くようにしてあります。このラベルは next タグで飛んでくる必要も、セーブデータ名を変える必要もないので *(半角アスタリスクと半角縦棒のみ) まで省略してしまえます。ラベル名を省略すると next タグでジャンプすることはできませんが、ロード時のジャンプでは使えるのでこれで大丈夫です。

■セーブデータの消去

セーブしたデータは、「 krkr.exe 」と同じフォルダの「 savedata 」フォルダに保存されています。このフォルダの中身を削除するとゲームのセーブデータを削除できます。「 krkr.exe 」と同じフォルダの「 clean.bat 」を実行すると savedata フォルダの中身を削除してくれます。

開発中に何かおかしい、と思ったらとりあえずセーブデータを削除してみると直るかもしれません。 KAGEX スクリプトのラベル名が変わったりすると、そもそもそのデータは使えなくなります。そうでなくとも、ゲームが終了する際には色々なデータが自動的に保存され、次にゲームを起動したときにそれらが読み込まれます。開発中で設定を変更したりしているとデータとのズレが生じて、設定の変更が反映されなかったり、動作がおかしくなったりする可能性があります。

ウィンドウ枠を引っ張ってウィンドウサイズを変更してからゲームを再起動してみると、データが保存されているというのがわかるとおもいます。再起動してもウィンドウサイズが変更された状態になっています。そこで「 clean.bat 」を実行してデータを削除し、もう 1 度ゲームを実行してみるとウィンドウサイズが戻っています。

大抵の場合はデータを削除しなくても大丈夫なのですが、何故か上手くいかないと

思ったらデータを削除してみてください。それで直ったらおめでとうございます。直らなかったらどこか別に問題があります。

■選択肢

```
*start| スタート
選択肢を使ってみます。
好きな方を選択してください。

@seladd text= 最初に戻る target=*start
@seladd text= 選択肢１ target=* 選択肢１
@seladd text= 選択肢２ target=* 選択肢２
@select

* 選択肢１
選択肢 1 が選ばれました。

@next target=*start

* 選択肢２
選択肢 2 が選ばれました

@next target=*start
```

選択肢は、「 seladd 」タグで選択肢を登録→「 select 」タグで登録した選択肢を表示、という流れになります。

「 seladd 」タグでは「 text 」属性に選択肢として表示する文字、「 target 」属性にその選択肢が選択されたときにジャンプする先のラベル名を指定します。 next タグと同じく「 storage 」属性を使うことで他のファイルのラベルへジャンプさせることもできます。

「 select 」タグでは、それまでに seladd タグで登録された選択肢を表示し、プレイヤーが選択肢を選択するまでスクリプトの実行を停止します。選択されたら、その選択肢で指定されているラベルからスクリプトの実行を再開します。

■空白を含む属性値

選択肢に「スペースのある 選択肢」と表示したいとします。単純に@seladd text= スペースのある 選択肢とすると、スペースの部分で属性の区切りとなってしまい、正しく動作しません。「text= スペースのある」「選択肢」という 2 つの属性と解釈されてしまいます。

このように属性値にスペースを含む値を指定したい場合、 "" (半角ダブルクォート) で囲むと正しく動作します。具体的には @seladd text=" スペースのある 選択肢2" のようになります。 seladd タグに限らず、どのタグでも同じように使えます。さらに言えば、スペースを含まない属性値を "" で囲んでもかまいません。@next storage="01.ks" target="*start" でもしっかり動作します。囲んだほうが見やすいのであれば囲んでください。本書では "" をタイプするのが面倒なので必要のない限り囲みません。

4. 画像操作スクリプト

3 章では画像の表示、非表示のやり方くらいしか説明していませんが、この章ではもう少し見た目を豪華にするスクリプトを紹介していきます。

■トランジションの基本

3 章では画像は瞬時に出たり消えたりしていました。大抵のデジタルノベルではじわじわとフェードインしたりフェードアウトしたりしています。それをやってみます。吉里吉里では「トランジション」と呼ばれる機能です。

```
*start| スタート
@ 街 昼 trans=crossfade time=2000
背景をフェードインしました。

@ しおり ポーズ a 制服 笑顔 前 trans=crossfade time=1000
【しおり】「立ち絵をフェードインしました。」

@ しおり 消 trans=crossfade time=500
立ち絵をフェードアウトしました。
```

トランジションをするには「 trans 」属性を使います。 trans 属性に「 crossfade 」を指定すると最も基本的なフェードイン、アウトである「クロスフェードトランジション」ができます。単純に、出てきたり消えたりします。

「 time 」属性でトランジションにかける時間をミリ秒 (ms) 単位で指定します。「 1 ミリ秒＝ 1 秒の 1000 分の 1 」です。time=1000とすると 1000ms ＝ 1 秒でトランジションします。time=500では 500ms ＝ 0.5 秒になります。

trans 属性を使うと、トランジションが終了するまでスクリプトの実行が一時停止されます。背景が完全に表示されてから 背景をフェードインしました。というテキストが表示されます。このような動作を「同期動作」といいます。

同期動作に対して「非同期動作」という概念があります。非同期動作はテキスト表示と同時進行でトランジションなどが実行される動作です。非同期動作でない動作が同期動作になります。とりあえず使ってみます。

```
*start| スタート
@ 街 昼  trans=crossfade time=2000 nosync
背景を非同期フェードインしています。

@ しおり ポーズ  a  制服 笑顔 前  trans=crossfade time=1000 nosync
【しおり】「立ち絵を非同期フェードインしています。」

@ しおり 消  trans=crossfade time=1000 sync
立ち絵を同期フェードアウトしました。

; 何度か試せるように最初にジャンプしています
@next target=*start
```

属性値を省略した「 nosync 」属性をつけると非同期動作、「 sync 」属性をつけると同期動作になります。背景や立ち絵のトランジションと同時に、テキストの表示も行われているのがわかるでしょうか。「テキスト設定」メニューの「テキスト速度」を遅くして試してみるとわかりやすいかもしれません。

最後の @ しおり 消 trans=crossfade time=1000 sync は sync 属性がついているので先ほどと同じ同期動作です。トランジションについてはデフォルトで同期動作となっているので「 sync 」属性はつけてもつけなくても変わりません。

非同期動作の実際の仕組みとしては、 @ 街 昼 trans=crossfade time=2000 nosync でトランジションが開始されたら、トランジションの終了を待たずにそのままスクリプトの実行を継続します。なので 背景を非同期フェードインしています がそのまま実行され、空行でのクリック待ちで一時停止します。

同期動作では、トランジションを開始したらそれが終了するまでスクリプトの実行が一時停止され、終わったら実行再開、となっています。なのでトランジションとテキストの表示は同時には行われません。

同期動作においてトランジションの終了を待っている間にクリックされると、トランジションを強制的にキャンセルしてスクリプトの実行を再開します。非同期動作の場合は、ページ送りの時点でまだトランジション中であれば強制的にキャンセルされます。トランジションが「キャンセル」されると瞬時に終了後の状態になります。フェードインなら瞬時に表示され、フェードアウトなら瞬時に消えることとなります。

```
*start| スタート
@ 街 昼 trans=crossfade time=15000 nosync
背景を 15 秒かけて非同期フェードインしています。

非同期動作の途中でページ送りが発生すると、非同期動作はキャンセルされます。

@ しおり ポーズ a 制服 笑顔 前 trans=crossfade time=15000 nosync nowait
【しおり】「 15 秒かけて非同期フェードインしています。」

nowait 属性をつけるとページ送りでもキャンセルされません

@ しおり stoptrans
強制的にトランジションをキャンセルしました。
```

「 nowait 」属性をつけた非同期トランジションはページ送りでキャンセルされなくなります。とはいっても、いつまでも実行されていても困るので「 stoptrans 」属性をつけると、その時点で nowait 属性がついた非同期トランジションもキャンセルできます。キャンセルされるのは「 stoptrans 」属性をつけた画像のみなので注意してください。 @ しおり stoptrans では、仮に背景が非同期トランジションしていたとしてもそちらはキャンセルされません。属性としてではなく、 @stoptrans とタグとして使えば全てのトランジションをキャンセルすることもできます。

```
*start| スタート
@begintrans
@ 街 昼
@ しおり ポーズ a 制服 笑顔 前 右
@endtrans trans=crossfade time=2000
背景と立ち絵を同時に同期トランジションしました。

@ かえで ポーズ a 私服 無表情 前 左 trans=crossfade time=1000 nosync
@ しおり ポーズ b trans=crossfade time=1000 nosync
立ち絵を 2 枚同時に非同期トランジションしています。

@begintrans
@ 黒
@ かえで 消
@ しおり 消
@endtrans trans=crossfade time=1000 sync
背景と立ち絵を同期トランジションで消去しました。
```

背景と立ち絵を同時にトランジションする際は「 begintrans 」タグと「 endtrans 」タグを使います。トランジションで変更したい内容を @begintrans と @endtrans の間に書きます。間に書いた画像関連の変更は、書いた時点では反映されず、 endtrans タグに指定したトランジションで一気に更新されることになります。 begintrans タグには属性がありません。 @endtrans trans=crossfade time=2000 のようにトランジションを指定することになります。

背景を含まず、立ち絵を 2 枚以上同時に「非同期」トランジションしたい場合は、begintrans と endtrans を使わずにそのまま並べてしまっても OK です。

```
*start| スタート
@begintrans
@ 街 昼
@ しおり ポーズ a 制服 笑顔 前 右
@endtrans trans=crossfade time=5000 sync
背景と立ち絵を同期トランジションで表示しました。

@begintrans
@ しおり ポーズ b
@ かえで ポーズ a 私服 無表情 前 左
@endtrans trans=crossfade time=2000
立ち絵を 2 枚同時に同期トランジションしました。

@ しおり 私服 怒り trans=crossfade time=1000 nosync
@ かえで 笑顔 trans=crossfade time=1000 nosync
@wt
立ち絵を 2 枚同時に同期トランジションしました。
```

背景を含まずに複数の立ち絵を同時に「同期」トランジションする場合は、begintrans タグと endtrans タグを使ってもいいですし、「 wt 」タグを使う方法もあります。 @wtでは同期 / 非同期に関わらず全てのトランジションが終了するまでスクリプトの実行を一時停止します。上のスクリプトの最後のように、非同期トランジションを開始してすぐに終了を待てば同期トランジションと同じ動作をさせることができます。

背景と立ち絵を同時にトランジションする場合は、バラバラにトランジションすると表示が変になってしまうので begintrans と endtrnas をつかってまとめてトランジションするようにします。

wt タグでは「 canskip 」属性が使えます。 true を指定するとクリックでトランジションをキャンセルできます。デフォルトの動作は canskip=trueです。 false を指定するとクリックしてもトランジションはキャンセルできず、トランジションが終了してからスクリプトの実行が再開されます。 sync 属性による同期動作では「 canskip 」属性は使えないので必ずクリックでキャンセルできます。

■その他のトランジション

trans 属性には crossfade 以外にも色々と指定できます。 1 つずつ紹介していきます。

サンプル素材フォルダの「 rule 」フォルダの「 rl うずまき .png 」と「 rl 右から左を、プロジェクトフォルダの「 rule 」フォルダにコピーしてください。

```
*start| スタート
@ 街 昼 trans=universal time=3000 rule=rl うずまき vague=0
背景を同期トランジションしました。

@ 街 夜 trans=universal time=3000 rule=rl うずまき vague=128 nosync
背景を非同期トランジションしています。

@begintrans
@ 向日葵 昼
@ しおり ポーズ a 私服 怒り 前 左
@layer name= お星様 file= 星 show
@endtrans trans=universal time=3000 rule=rl 右から左 vague=64
【しおり】「同期トランジションしました。」
```

trans 属性に「 universal 」を指定すると「ユニバーサルトランジション」になります。time 属性にはトランジションの時間を指定します。「 rule 」属性にはモノクロのグレースケール画像のファイル名を指定します。ユニバーサルトランジションでは rule 属性に指定された画像 (「ルール画像」と言います) の黒い部分から順に入れ替わっていき、白いところが最後に入れ替わります。「 vague 」属性はあいまい領域値というもので、大きな値を指定するほど入れ替わりの境界線がぼやけてなめらかなトランジションになります。

言葉で言ってもよくわからないので数値を変えて試してみるのがいいと思います。vague 属性は省略でき、その場合はvague=64を指定したとみなされます。

ルール画像は「 rl 」から始まるファイル名で「 rule 」フォルダに入っている必要があります。

```
*start| スタート
@ 向日葵 昼  trans=scroll time=3000 from="top" stay="nostay"
背景を同期トランジションしました。

@ 向日葵 夕  trans=scroll time=3000 from="left" stay="stayback" nosync
背景を非同期トランジションしています。

@ev 発見  trans=scroll time=3000 from="bottom" stay="stayfore"
【しおり】「同期トランジションしました。」
```

trans 属性に「 scroll 」を指定すると、「スクロールトランジション」になります。画像がスクロール（ スライド ）して切り替わるトランジションです。 time 属性にはトランジションの時間を指定します。「 from 」属性は、どちらから変更後の画像がスライドしてくるかを指定します。「 left 」「 top 」「 right 」「 bottom 」のいずれかを指定それぞれ左、上、右、下からスライドしてきます。「 stay 」属性には「 nostay 」「 stayback 」「 stayfore 」のいずれかを指定します。「 nostay 」では、切り替わる前と後の画像のどちらもスライドします。ちょうど切り替わる前の画像が後の画像に押し出されるようにスライドされます。「 stayback 」では切り替わる前の画像がスライドして出ていき、その下から切り替わった後の画像が出てきます。「 stayfore 」では、切り替わった後の画像がスライドして出てきて、切り替わる前の画像にかぶさってきます。こちらもよくわからないので実際に試してみることをお勧めします。

```
*start| スタート
@begintrans
@ 向日葵 昼
@ しおり ポーズ a 私服 驚く 奥 左
@endtrans fade=2000
【しおり】「背景と立ち絵を同期トランジションしました。」

@ しおり 消  fade=1000 nosync
立ち絵を非同期トランジションしています。
```

trans=crossfade については よく使うので、簡単に「 fade 」属性を使うこともできます。属性値には trans=crossfade では time 属性に指定していたトランジションの時間を指定します。

■同期 / 非同期トランジションについて

テキスト表示と同時に別のこともする非同期動作は比較的重い（ 遅い ）処理です。そもそもトランジションはそれ単体でも重い処理です。近頃のパソコンなら問題ありませんが、あまり同時かつ大量に使うと表示がカクカクしたりするかもしれません。

昔のゲームではパソコンの性能の問題でほぼ同期トランジションが使われていました。

非同期動作のいいところはプレイヤーを待たせないところです。同期トランジションでは画像が切り替わる間にテキストが進むまでプレイヤーを待たせてしまいます。クリックすれば飛ばせますがそれすら面倒な人もいます。稀にクリックしても飛ばせないゲームもありますが、立ち絵の表情が変わるごとに待たされるようなゲームはクソゲーです。内容に関係なく開始 30 秒でゴミ箱行きです。 wt タグの canskip 属性は非常に紹介したくなかったのですが、表現の幅を狭めるのもよろしくないので紹介しました。使わないであげてください。切り替えにトランジションを使わない、というのもアリかもしれません。ここぞという場面でのみ使った方が効果的……かはわかりませんが。

現状、立ち絵の表情変更では非同期トランジション、背景ごと変わる場合は同期トランジション、という使い方がメジャーなように見えます。非同期トランジションにすると立ち絵なんか見なくなってしまう、という副作用もあるのですが、表情差分の種類は増えてる傾向にあります。いや最近は落ち着いているかもしれないです。増やしても誰が見ているんだろうと結構疑問だったりします。ボイスを最後まで聞く人なら立ち絵にも目がいくんでしょうか。ボイスもボイスカットや多重再生などの機能がついてどんどん飛ばせる方向になってますね。表情を見せるためか、テキストの横にフェイスウィンドウを出しているゲームもあります。

■不透明度

```
*start| スタート
@ リビング 昼
@ しおり ポーズ b 私服 驚く 前 中
背景とキャラクタを表示しました。

@ しおり opacity=128
【しおり】「不透明度を半分にしました。」

@ しおり opacity=0
【しおり】「不透明度を 0 にしました。」

@ しおり opacity=255
【しおり】「不透明度を 255 に戻しました」
```

「 opacity 」属性を使うとレイヤの不透明度を０～２５５まで変更できます。不透明度が低いと画像が半透明になり、後ろのレイヤの画像が透けて見える用になりま

す。高いと不透明になるため後ろのレイヤの画像は見えません。デフォルトでは最大値の 255 になっているため、立ち絵の後ろの背景は完全に隠れています。@ しおり opacity=128とすると不透明度が半分になるため後ろの背景が透けてみえるようになります。@ しおり opacity=0 すると、完全に透明になり立ち絵が見えなくなってしまいます。@ しおり opacity=255 で不透明度を 255 に戻すとまた立ち絵が見えるようになります。opacity 属性は立ち絵だけでなく、背景、イベント絵、環境レイヤについても同じように使用できます。

■time 属性によるアクション

アクションを使うと、属性の値を連続的に変化させることができます。単純に属性に値を指定するだけでは瞬時に切り替わってしまいますが、アクションを使うと動きのある表現が実現できます。

```
*start| スタート
@ リビング 昼
@ しおり ポーズ a 私服 驚く 奥 左
背景とキャラクタを表示しました。

@ しおり xpos=200 time=1000
【しおり】「 xpos の値を１秒間で変更しています。」

@ リビング xpos=-50 ypos=300 time=2000
背景の xpos と ypos の値を２秒間で変更しています。

@ しおり opacity=0 time=3000
【しおり】「３秒間で立ち絵を透明にしています。」
```

属性の値を変更する際に time 属性でミリ秒単位の時間を指定すると、属性の現在の値から指定された値まで、指定した時間をかけて変化するようになります。これを「アクション」といいます。

@ しおり xpos=200 time=1000では、 xpos 属性の値が現在の 0 から指定された 200 まで１秒間で変化します。 xpos 属性の値が「 0,1,2,3,..,199,200 」と変化していくので、見た目としては立ち絵が少しずつ右に移動していきます。 @ リビング xpos=-50 ypos=300 time=2000のように xpos 属性と ypos 属性を同時に指定すると、どちらも２秒間で変化します。 xpos 、 ypos 属性だけでなく、 opacity 属性についても同じように使用できます。

```
*start| スタート
@ リビング 昼
@layer name= お星様 file= 星 show
背景と環境レイヤを表示しました。

@ お星様 xpos=-200 time=1000 sync
環境レイヤを同期アクションで移動しました。

@ リビング xpos=100 ypos=-100 time=2000 sync
背景を同期アクションで移動しました。

@ お星様 opacity=0 time=3000 nosync
環境レイヤを非同期アクションで透明にしています。
```

トランジションと異なり、アクションはデフォルトで非同期動作になっています。デフォルト値が違うだけで、動作の仕組みはトランジションの同期 / 非同期動作と同じです。同期動作の場合はアクションが終了するまでスクリプトの実行が中断されます。「 sync 」「 nosync 」属性でそれぞれ指定できます。非同期動作はページ送りでキャンセルされます。

```
*start| スタート
@ リビング 昼
@layer name= お星様 file= 星 xpos=-200 ypos=-200 show
背景と環境レイヤを表示しました。

@ お星様 xpos=200 ypos=200 time=10000 nowait
環境レイヤを非同期アクションで移動しています。

nowait 属性をつけているのでクリックしてもキャンセルされません。

@ お星様 stopaction
強制的にキャンセルしました。
```

「 nowait 」属性もトランジション同様に使えます。トランジションをキャンセルするのは「 stoptrans 」属性でしたが、アクションは「 stopaction 」属性でキャンセルできます。 @stopactionとタグとして使うと全てのアクションをキャンセルすることもできます。

```
*start| スタート
@ 街 夕
@ かえで ポーズ a 私服 無表情 奥
背景と立ち絵を表示しました。

@ 街 xpos=-200 time=2000
@ かえで 右 time=2000
背景と立ち絵を非同期アクションで移動しています。

@ 街 xpos=200 time=10000 nosync
@ かえで 左 time=10000 nosync
@wact
街と立ち絵を同期アクションで移動しました。
```

背景と立ち絵など、複数の画像に同時に非同期アクションを使う場合はそのままタグを並べれば OK です。トラジションにおける begintrans タグと endtrans タグのようなタグはありません。

同期アクションを同時に使う場合は「 wact 」タグが使えます。 wt タグではトランジションの終了を待ちましたが、 wact タグではアクションの終了を待てます。非同期アクションを同時に実行して、 @wactでまとめてアクションが終了するのを待ちます。 canskip 属性でスキップできないようにすることも可能です。

```
*start| スタート
@ 街 昼 fade=1000 nosync
@ 街 xpos=-200 nosync time=2000 nosync
@wat
背景のトランジションとアクションを同時に実行しました。
```

wt タグではトランジション、 wact タグではアクションの終了を待ちますが、「 wat 」タグではトランジションとアクションの両方の終了を待ちます。 canskip 属性についても同じく使用できます。

wt 、 wact 、 wat タグは、そのタグの時点でトランジションやアクションが実行されている場合に、その全てが終了するまでスクリプトの実行を停止します。 canskip が省略されるか、canskip=falseの場合にクリックされれば、実行されているトランジション / アクションを全てキャンセルします。

これらのタグの時点で 1 つも実行されていない場合は、そこで停止せずに単純に無視されます。

```
*start| スタート
@ 街 昼 fade=500
背景を表示しました。

@ 街 xpos=200 ypos=-200 opacity=0 time=2000
背景を非同期移動しながら透明にしています。

@ 街 xpos=0 ypos=0 time=2000
@ 街 opacity=255 time=4000
背景を違う時間で非同期移動しながら不透明にしています。
```

xpos 、 ypos 属性と opacity 属性に対して同時にアクションを使う事もできます。@街 xpos=200 ypos=-200 opacity=0 time=2000 のように１つのタグに書いた場合は time 属性による時間指定が共通になりますが、タグを２つにわければ異なる時間を指定することも可能です。ただし xpos と ypos については特別で、これらを別々にすることはできません。

```
*start| スタート
@ 街 昼
@layer name= 星１ file= 星 show ypos=100
@layer name= 星２ file= 星 show ypos=0
@layer name= 星３ file= 星 show ypos=-100
背景と環境レイヤを表示しました。

@ 星１ xpos=300 time=2000 accel=accel
@ 星２ xpos=300 time=2000 accel=const
@ 星３ xpos=300 time=2000 accel=decel
星１を加速移動、星２を等速移動、星３を減速移動しています。

@ 星１ xpos=-300 time=4000 accel=acdec
@ 星２ xpos=-300 time=4000 accel=const
@ 星３ xpos=-300 time=4000 accel=accos
星１を加減速移動、星２を等速移動、星３をコサイン加減速移動しています。
```

復習ですが ypos 属性は数値が大きいほど表示位置としては上なので、実行すると星１が一番上、星２が真ん中、星３が一番下に表示されます。

「 accel 」属性を使うことで、属性の値の変化を加速させたり減速させたりすることができます。アクション全体でかかる時間は time 属性に指定した値で変わりません。属性値に「 accel 」を指定すると最初はゆっくり、後から急激に値が変化するように

なります。「 decel 」を指定すると最初は急激に、後からゆっくりと変化するようになります。「 const 」を指定すると「 accel 」属性を使っていない場合と変わらず、一定の速さで値が変化します。

「 acdec 」と「 accos 」は加減速で、前半は加速、後半は減速するようになります。実行してみるとわかりますがほとんど変わりません。「 accos 」の速度変化の方がわずかに緩やかになります。

■path 属性によるアクション

```
*start| スタート
@ 街昼
@layer name= 星１ file= 星 show
背景と環境レイヤを表示しました。

@ 星１ path="(200,0,0),(200,200,255),(0,0,128)" time=4000
移動しています。

@ 星１ path="(-200,0,255),(-200,200,255)" time=4000 spline
スプライン移動しています。
```

「 path 」属性を使うと複数点の移動と不透明度のアクションができます。属性値は「 (xpos の値 ,ypos の値 ,opacity の値) 」というセットをコンマでつないで、いくつか並べた形になります。 @ 星１ path="(200,0,0),(200,200,255),(0,0,128)" time=4000 は、「xpos=200,ypos=0,opacity=0」「xpos=200 、 ypos=200 、 oopacity=255 「xpos=0 、 ypos=0 、 opacity=128」 が３つセットになっています。 time 属性に指定した時間でその３点を順番に移動していきます。位置のみを変化させて、不透明度を変化させたくない場合は、opacity の部分を全て 255 など一定の値にしておきます。

path 属性とともに「 spline 」属性を使うとスプライン補間された経路で各点を移動します。各点をなめらかにつないで移動させることができます。他に sync 、 nosync 、 nowait 属性使えますが、「 accel 」属性は使えません。

■画像の回転

```
*start| スタート
@ 街 昼
背景を表示しました。

@ 街　rotate=180
背景の角度を１８０度にしました。

@ 街　rotate=0
角度をもどしました。
```

「 rotate 」属性を使うと画像を回転できます。属性値には反時計回りで何度の角度に回転させるかを指定します。マイナスの値を指定すると時計回りの角度になります。背景だけでなく立ち絵やイベント絵、環境レイヤも同じく回転できます。

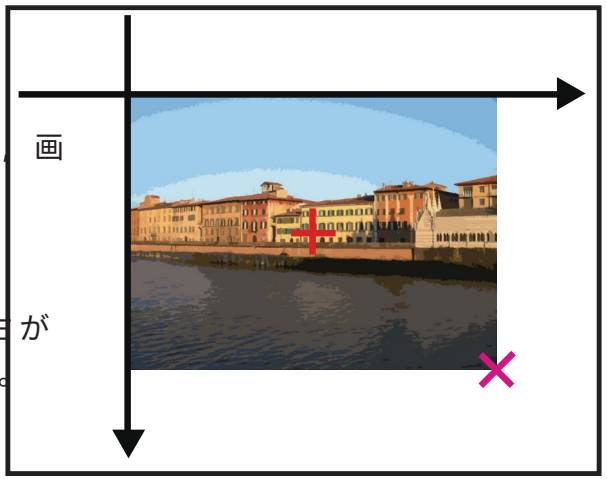
```
*start| スタート
@ リビング 昼
背景を表示しました。

@ リビング　rotate=360 time=3000
背景を反時計回りに１回転させています。

@ リビング　rotate=-360 time=6000 sync accel=acdec
背景を時計回りに２回転させました。
```

time 属性によるアクションも使えます。 rotate=360 time=3000では、アクションによってデフォルトの 0 から、指定された 360 まで「 0,1,2,3...359,360 」のように値が変化していくので、見た目としては反時計回りに 1 回転することになります。次の rotate=-360 time=6000では現在の値の 360 から指定された -360 まで「 360,359,358,..,1,0,-1,...,-359,360 」と rotate の値が変化していきます。見た目としては時計回りに 2 回転することになります。 sync 、 nosync 、 accel 属性も問題なく使えます。

■回転原点

```
*start| スタート
@ 街 昼
背景を表示しました。

@ 街 rotate=360 time=1000
背景を反時計回りに 1 回転しています。

@ 街 afx=0 afy=0
回転原点を画像左上に変更しました。

@ 街 rotate=720 time=1000
背景を反時計回りに 1 回転しています。
```

「 afx 」属性と「 afy 」属性で「回転原点」を指定できます。回転原点とは、回転の際に中心となる点 (座標) で、その点の表示位置が回転の際にずれないように回転します。



回転原点は画像上の点で、画像左上を (0,0) 、画像右下を (画像縦幅 , 画像横幅) とした直交座標系で指定します。右の画像を見てください。背景画像のサイズは 800 × 600 です。横が X(afx), 縦が Y(afy) で、 afx は右がプラス、 afy は下がプラスになります。前述の通り画像左上が (0,0) で、この背景画像のサイズは 800 × 600 のため画像右下の × の点が (800,600) になります。画像中心の＋の点は (400,300) です。

背景のデフォルトの回転原点は画像中央になっています。 800 × 600 の画像では (400,300) です。なので初めの @ 街 rotate=360 time=1000 は、右の画像の様に中央を中心として回転が実行されます。

次に @ 街 afx=0 afy=0で回転原点を画像左上の (0,0) に変更しています。このとき表示位置が右下の方にずれてしまいますが、これについては後述します。とりあえず次の @ 街 rotate=720 time=1000では、右の画像のように画像左上を中心に回転します。すでに 360 度まで回転しているので、もう 1 回転させるために 720 度を指定しています。

```
*start| スタート
; 回転が見やすいように位置を上にずらしています
@ しおり 制服 ポーズ b 悲しみ 奥 中 ypos=300
立ち絵を表示しました

@ しおり rotate=180 time=2000
立ち絵を反時計回りに 180 度回転しています。

@ しおり afx=center afy=center
回転原点を中心にしました。

@ しおり rotate=0 time=2000
立ち絵を時計回りに 180 度回転しています。
```

立ち絵のデフォルトの回転原点は、画像中央ではなく画像下端中央になっています。なので回転は画像下端を中心にして実行されます。

また、 afx 、 afy 属性では、数値で座標を指定する代わりにせずに文字で指定することもできます。 afx 属性に「 center 」を指定すると、回転原点の X 座標が画像中央の点の X 座標に一致します。 afy 属性に「 center 」を指定すると、回転原点の Y 座標が画像中央の点の Y 座標に一致します。よって、afx=center afy=centerとすると画像中央が回転原点となります。また、 afx 属性には「 left 」または「 right 」を指定することもできます。 afx=leftの場合は画像の左端、 afx=rightの場合は画像の右端が回転原点になります。 afy 属性には「 top 」または「 bottom 」が指定でき、それぞれ画像上端、画像下端が回転原点になります。これでいうと、立ち絵のデフォルトの回転原点は

afx=center afy=bottomとなります。

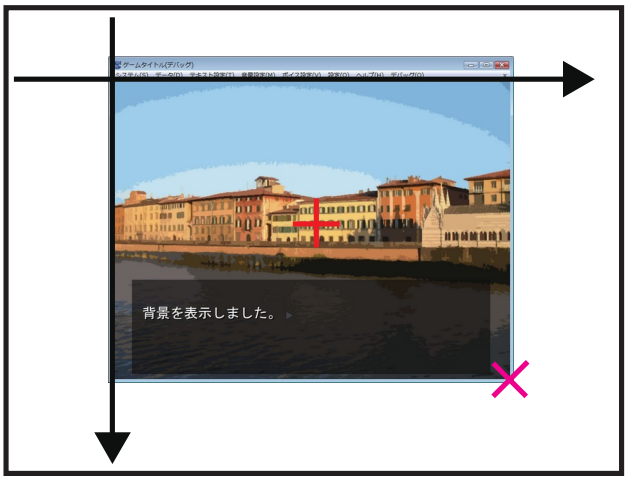
背景、イベント絵、環境レイヤのデフォルトの回転原点は画像中央、つまりafx=center afy=centerになっています。立ち絵のみ異なっているのは、立ち絵の足元の位置が回転した際にずれないようにするためです。サンプルではそこまで描かれていませんが、足元まで含めて全身が描かれている立ち絵画像を使う場合、画像下端中奥はちょうど足元になります。立ち絵は一応地面に立っているはずなので接地面の足元がずれなければ都合のいいことが多いはずです。

■表示原点

```
*start| スタート
; 回転が見やすいように位置を上にずらしています
@ しおり 制服 ポーズ b 悲しみ 奥 中 ypos=300
立ち絵を表示しました

@ しおり rotate=180 time=2000
立ち絵を反時計回りに 180 度回転しています。

@ しおり afx=center afy=center
回転原点を中心にしました。

@ しおり rotate=0 time=2000
立ち絵を時計回りに 180 度回転しています。
```

回転原点の他に、「表示原点」があります。回転原点は回転の中心となる画像上の座標でしたが、表示原点は表示の中心となるウィンドウ上の座標になります。ウィンドウ左上が (0,0) 、ウィンドウ右下が (ウィンドウ幅 , ウィンドウ高さ) となる直交座標系で指定されます。

右の画像を見てください。回転原点とほぼ同じですが、画像上ではなくウィンドウ上の座標であることに注意してください。ウィンドウサイズも 800 × 600 なのでウィンドウ右下は (800,600) 、ウィンドウ中央は (400,300) になっています。



画像は表示原点を中心として表示されます。つまり、 xpos=0 ypos=0のときに回転原点と表示原点が重なるように表示されます。

@ 街 昼で背景を表示すると、回転原点 (画像中央) がデフォルトの表示原点であるウィンドウ中央と重なる位置に表示されます。 xpos=100とすればそこから右に 100px ずれ、 ypos=100とすれば上に 100px ずれる、という仕組みになっています。

@ 街 orx=0 ory=0と表示原点をウィンドウ左上に変更すると、回転原点 (画像中央) が表示原点 (ウィンドウ左上) に重なるように画像の位置が変更されます。逆に回転原点が @ 街 afx=left afy=topと変更されても、それに伴って画像位置が変更されます。ここでは回転原点である画像左上が表示原点であるウィンドウ左上と重なるため、結果としては最初の表示と同じになります。

回転原点についても表示原点と同じように orx 属性は「 left 」「 center 」「 top 」(それぞれウィンドウ左端、中央、右端の X 座標) 、 ory 属性は「 top 」「 center 」「 それぞれウィンドウ上端、中央、下端の Y 座標) で指定できます。

また、回転原点はウィンドウ上の座標ではありますが、画像ごとに別々に設定されます。ウィンドウに 1 つしかない、という事ではありません。

回転原点は背景、立ち絵、イベント絵、環境レイヤのすべてでウィンドウ中央がデフォルトとなっています。

3 章で立ち絵を初めて表示したとき、立ち絵の位置が上にずれていました。それは回転原点と表示原点の機能によって、画像下端が画面中央となるような位置になっていたためです。その位置ではまずいので、立ち絵については特別に縦位置のオフセット (yoffset) を指定できるようになっていた、ということになります。オフセットに指定されたピクセル数だけ表示原点の位置がずれたような動作になります。

■拡大率

```
*start| スタート
@ 街 昼
背景を表示しました。

@ 街 zoomx=50
背景の幅を 50% に縮小しました。

@ 街 zoomy=200
背景の高さを 200% に拡大しました。

@ 街 zoom=100
背景の拡大率を元に戻しました。
```

「 zoomx 」「 zoomy 」属性に画像の拡大率を指定することができます。それぞれ横方向、縦方向の拡大率で、デフォルトの 100 が画像のサイズそのままとなっています。 50 を指定すると通常の半分、 200 を指定すると通常の倍、のようになっています。「 zoom 」属性を使って zoomx 、 zoomy 属性に一括で同じ値を指定することもできます。

```
*start| スタート
@ 街 昼
@ しおり 前 制服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@ 街 zoom=50 time=2000 sync
背景を 50% に縮小しました。

@ しおり zoom=200 time=2000
立ち絵を 200% に拡大しています。
```

zoom 、 zoomx 、 zoomy 属性についても time 属性によるアクションが使えます。rotate 属性での回転と同様に、拡大縮小も回転原点を中心として実行されます。つまり、回転原点の表示位置は拡大、縮小されてもずれません。

```
*start| スタート
@ 街 昼
@ しおり 前 制服 ポーズ b 無表情
背景と立ち絵を表示しました。

@ 街 zoom=-100 time=2000
背景を -100% に拡大しています。

@ しおり zoomx=-100 time=2000
立ち絵の幅を -100% に拡大しています。

@ 街 zoom=-200 time=2000
背景を -200% に拡大しています。
```

zoom 、 zoomx 、 zoomy 属性にマイナスの値を指定すると回転原点を中心として反転します。「 zoomx 」の場合は左右に反転、「 zoomy 」の場合は上下に反転、「 zoom なら上下左右が反転した見た目になります。 -200 を指定すれば反転した上で 2 倍に拡大されます。

■反転

```
*start| スタート
@ 街 夕
@ しおり 奥 私服 ポーズ b 無表情
背景と立ち絵を表示しました。

@ 街 flipy
背景を縦方向に反転しました。

@ しおり flipx
立ち絵を横方向に反転しました。

@ 街 flipx
背景を横方向にも反転しました。

@ 街 flipx=false flipy=false
背景の反転を戻しました。
```

「 flipx 」「 flipy 」属性を使って画像を反転することもできます。 true を指定すると反転、 false を指定すると反転を元に戻します。属性値の true は省略できることを覚えているでしょうか。 flipx 、 flipy による反転は回転原点に関係なく、画像の表示位置

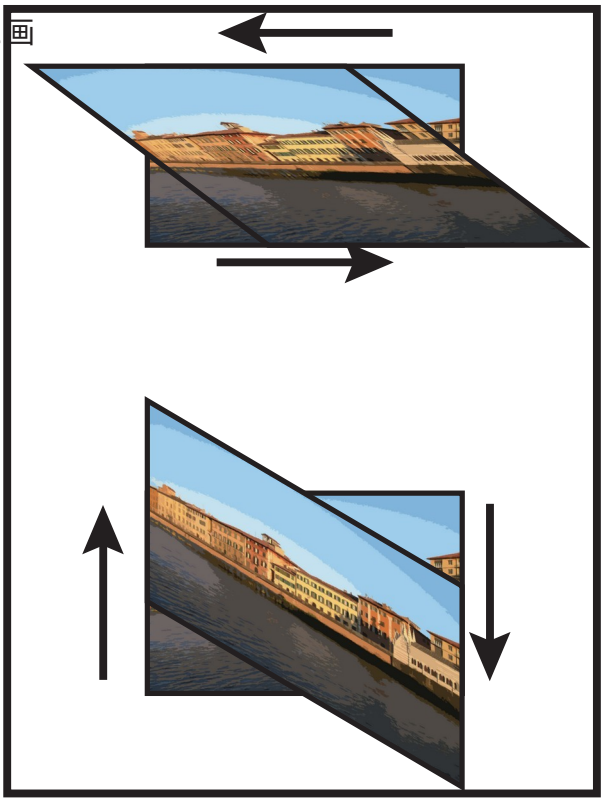
がずれないように反転されます。手軽に使えるので反転する場合は「 zoom 」より便利かもしれません。

■傾斜率

```
*start| スタート
@ 街 昼
@ しおり 前 制服 ポーズ b 無表情
背景と立ち絵を表示しました。

@ 街 slantx=100
背景を横方向 100 に傾斜しました。

@ しおり slantx=-100
立ち絵を横方向 -100 に傾斜しました。

@ 街 slantx=0 slanty=100
背景を縦方向 100 に傾斜しました。
```

「 slantx 」「 slanty 」属性を使うと画像を傾斜させることができます。どちらも 0 がデフォルトで全く傾斜していない状態です。

傾斜させると四角形が平行四辺形となるように変形されます。 100 のとき鋭角 45 度の平行四辺形になります。言葉で上手く説明できないので右の画像をみてください。値が大きいほど傾斜がきつくなります。マイナスの値では逆方向に傾斜します。傾斜も回転拡縮と同じく、回転原点が中心となり、その位置がずれないように傾斜されます。



```
*start| スタート
@ 街 昼
背景を表示しました。

@ 街 slantx=100 time=2000
背景を横方向 100 に傾斜しています。

@ 街 slantx=0 time=2000 sync
背景の傾斜をもどしました。

@ 街 slanty=100 time=2000
背景を縦方向 -100 に傾斜しています。
```

傾斜でも time 属性によるアクションが使えます。注意点も特にありません。「 slantx 」と「 slanty 」を同時にアクションで使うこともできます。上のスクリプトではよくわからなくなるので１つずつ使っています。

■ぼかし

```
*start| スタート
@ 街 昼
背景を表示しました。

@ 街 slantx=100 time=2000
背景を横方向 100 に傾斜しています。

@ 街 slantx=0 time=2000 sync
背景の傾斜をもどしました。

@ 街 slanty=100 time=2000
背景を縦方向 -100 に傾斜しています。
```

「 blurx 」「 blury 」属性を使うと画像をぼかすことができます。どちらも 0 がデフォルトで全くぼかしをかけていない状態です。 blurx 属性が横方向、 blury 属性が縦方向のぼかしです。画像が変形されるわけではないので回転原点は関係ありません。
「 blur 」属性で blurx 、 blury 属性に同じ値に指定することもできます。

```
*start| スタート
@ しおり 前 私服 ポーズ a 笑顔
立ち絵を表示しました。

@ しおり blurx=50 time=1000
立ち絵を横方向にぼかしています。

@ しおり blurx=0 blury=50 time=1000 accel=accel
立ち絵を縦方向にぼかしました。

@ しおり blur=0 time=1000 sync
ぼかしを消しました。

@ しおり allclear
全クリアしました。
```

ぼかしでも time 属性によるアクションも使えます。

大きな値でぼかしを使うと、ぼかしを 0 にしても周りに色が残ってしまうことがあります。上のサンプルでもぼかしを消した時点で周りに色が残ってしまいます。ぼかしに限らず回転や拡縮、移動などでも急激なアクションを使うと残ってしまったりします。これが起こるのは、もともとの画像サイズよりも広い範囲に色が描画された場合です。ぼかしで大きな値を入れると本来の画像のサイズ外にも色が塗られるので、その部分が残ってしまいます。

回避策としては、画面外にはみ出すような大きな値を使わないか、画像サイズ自体を大きくするかです。画像の周りに透明な部分を付け足せば画像サイズは大きくできます。

「 allclear 」属性を使うと画像サイズの外の部分に残っている色も強制的に綺麗にしてくれます。すぐに「 allclear 」を使うことができる場合はこの属性を使えばあまり目立たなくできるかもしれません。上のスクリプトでは、ぼかしを消してからプレイヤーが次のテキストに進んで「 @ しおり allclear 」が実行されるまで色が残ったままになってしまいます。

■ラスタースクロール

```
*start| スタート
@ 街 昼
@ しおり 奥 私服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@ 街 raster=100
背景をラスタースクロールしています

@ 街 raster=0
停止しました。

@ 街 raster=100 rasterlines=200 rastercycle=2000
ラスタースクロールしています。

@ 街 raster=200
スクロールする横幅を変更しました。

@ 街 raster=200 rasterlines=400
ラインの縦幅を変更しました。

@ 街 raster=200 rastercycle=10000
ラスターの時間を変更しました。
```

「 raster 」属性では横ラスタースクロールが使えます。画像が横方向にグニャグニャとスクロールします。

raster 属性には、横方向に最大でどれだけスクロールさせるかをピクセル単位で指定します。raster=0とすればラスタースクロールを停止できます。

「 rasterlines 」属性には縦方向のラインの幅をピクセル単位で指定します。この属性に指定したピクセル数ごとに左右どちらにスクロールされるかが変わります。デフォルト値は 100 になっています。「 rastercycle 」属性には左右 1 往復スクロールするのにかかる時間をミリ秒単位で指定します。デフォルト値は 1000 です。

rasterlines 属性または rastercycle 属性を指定する場合は必ず「 raster 」も同時に指定する必要があります。@街 rasterlines=400ではなく @ 街 raster=200 rasterlines=400とする必要があります。

```
*start| スタート
@ 街 昼
@ しおり 奥 私服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@ 街 raster=100 time=2000
背景をラスタースクロールしています。

@ 街 raster=0 time=2000
停止しています。
```

raster 属性については time 属性によるうアクションが使えます。 @ 街 raster=100 time=2000では raster 属性の値が「 0,1,2,...,99,100 」と変化していくので徐々にスクロールされる横幅が広くなっていきます。 @ 街 raster=0 time=2000はその逆で、 2 秒かけてラスタースクロールが停止します。

rasterline 属性と rastercycle ゾック製にはアクションは使えません。 time 属性に関係なく、すぐに指定された値に変更されます。

ここまで、サンプルスクリプトでは背景と立ち絵くらいしか使っていませんが、回転、拡大率、反転、傾斜度、ぼかし、ラスタースクロールは背景、立ち絵、イベント絵、環境レイヤで同じように使えます。

■相対指定

```
*start| スタート
@ 街 昼
@ しおり 奥 私服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@ しおり xpos=@100
【しおり】「右に 100px 移動しました。」

@ しおり xpos=@100
【しおり】「もう 1 度右に 100px 移動しました」

@ しおり xpos=@-400
【しおり】「左に 400px 移動しました。」
```

属性値の先頭に @(半角アットマーク) をつけると、現在の値からの相対指定になります。その属性の現在の値に、指定した値を足した値が実際の値になります。

xpos はデフォルトで 0 なので、最初の@ しおり xpos=@100では xpos の値が 0+100=100 になります。次の@ しおり xpos=@100では、この時点で xpos の値は 100 になっているので 100+100=200 になります。最後の @ しおり xpos=@-400では、 200+(-400)=-200 になります。

相対指定は time 属性によるアクションが使える属性であれば使用できます。ここで time 属性によるアクションが使える属性をまとめておくと、「 xpos 」「 ypos 」「 opacity 」「 rotate 」「 zoomx 」「 zoomy 」「 zoom 」「 slantx 」「 slanty 」「 raster 」となります。

■パーセント指定

```
*start| スタート
@ 街 昼
@ しおり 奥 私服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@ しおり xpos=100%
【しおり】「右 100% の位置に移動しました。」

@ しおり xpos=0% ypos=50%
【しおり】「上 50% の位置に移動しました。」

@ しおり rotate=100% time=2000
【しおり】「反時計回りに一回転しています。」

@ しおり opacity=50%
【しおり】「不透明度を半分にしました。」
```

属性値の末尾に %(半角パーセント) をつけるとパーセント指定になります。パーセント指定が使える属性は相対指定が使える属性と同じです。

zoomx 、 zoomy 、 zoom 、 slantx 、 slanty 属性については元から率による指定なので % をつけてもつけなくても変わりません。

ピクセル単位の指定である xpos 、 ypos 、 blurx 、 blury 、 blur 、 raster について 0% が 0 、 100% が x では画面横幅、 y では画面縦幅の半分のピクセル数になります。デフォルトの画面横幅は 800 なので@ しおり xpos=100%では 800 の半分の 400 だけ右にずれます。次の ypos=50%では、デフォルト画面縦幅 600 の半分の 300 の 50% で、150 だけ上にずれることになります。xpos=-50%のようにマイナスをつけて左にずら

すこともできます。

rotate 属性では 1 回転の 360 が 100% に対応します。 oapcity 属性では最大値の 255 が 100% になります。

■初期値指定

```
*start| スタート
@ 街 昼
@ しおり 奥 私服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@ しおり xpos=-100:100 time=2000
【しおり】「 xpos=100 から xpos=-100 に移動しています。」

@ しおり zoom=100:200 time=2000
【しおり】「 200% から 100% に拡大しています。」

@ しおり pos= 中 : 右外 time=2000
; @ しおり 中 : 右外 time=2000 でも OK
【しおり】「右外から中に移動しています。」
```

time 属性によるアクションでは、現在の値から指定された値まで属性が変化します。属性値の後ろに「 : 初期値」の形でくっつけて、アクションを始める最初の値を指定することができます。

@ しおり xpos=-100:100 time=2000 すると、 xpos 属性が「 100,99,98,...,-99,-100 」のように変化することになります。次の @ しおり zoom=100:200 time=2000 も同様に「 200,199,198,...,101,100 」と変化していきます。立ち絵の pos= 中 : 右外 については属性名を省略して 中 : 右外 のように使うこともできます。

初期値指定が使える属性は、初期値指定と同じく time 属性によるアクションが使える属性全てとなります。

相対指定、パーセント指定、初期値指定を組み合わせて使うこともできます。 xpos=100:@-100 や xpos=@-40%:@20 のようにｚなりますが、必要になったことも無いので特に説明しません。

■回転原点のアクション

```
*start|  スタート
@ 街 昼
@ しおり 奥 私服 ポーズ  a  笑顔
背景と立ち絵を表示しました。

@ 街  afx=0 afy=0 time=2000
背景の回転原点を変更しています。

@ 街  afx=center afy=center rotate=180 time=2000
回転原点を変更しながら回転しています。
```

回転原点についてもアクションが使えます。 @ 街 afx=0 afy=0 time=2000では、回転原点の位置が変わるにつれて画像の表示位置も変わります。 @ 街 afx=0 afy=0 rotate=180 time=2000ではそれに加えて画像の回転も行われます。

他のアクションを使える属性と異なり、 afx 、 afy 属性のアクションでは相対指定、パーセント指定、初期値指定は使えません。

回転原点まで使うとそこそこ複雑な動きもできるようになりますが、実際に複雑な動きをさせる場合はアクションを定義して使うことが多いです。アクション定義の仕方は後ほど紹介します。

■単語登録

余白を埋めるためにまた脇道にそれますが、 IME の単語登録はなかなか便利です。何それ? という人は検索してみてください。

@ しおなんて毎回真面目に打つのは結構だるいです。今はｑｓで @ しおり ｑｋで @ かえでと変換できるようになっています。ｑｓ、ｑｋに特に意味はありません。タイトルごとによく使う文字列はどんどん登録してしまっています。

共通で使っているものとしては、例えばｓの変換が半角スペースになっています。属性の間のスペースを打つためにいちいち半角 / 全角切り替えなんてやめましょう。これは 5 章のボイスの解説で出てきますが、ｐｄｌｄでplayrun= ラベル化変換できます。よく使うのに属性名が長くて大変なんですね。

私のやり方を真似する必要は無いのですが、まあこんな省力化の方法もあるので参考にしてください。

■カメラ操作

KAGEX には「カメラ」という概念があります。「 stage 」「 char 」「 event 」「 lay タグによって表示される画像は、仮想的なカメラを通して画面にうつされている、と考えます。

カメラ機能は一応紹介しますが、使い方が面倒なので実際はそれほど使われなかったりします。

```
*start| スタート
@ 街 昼
@ しおり 前 右 制服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@env camerax=50
カメラ位置を右に 50 ずらしました。

@env cameray=50
カメラ位置を上に 50 ずらしました。

@env camerazoom=200
カメラの拡大率を 200% にしました。

@env camerarotate=45
カメラを反時計回りに 45 度傾けました。
```

カメラは「 env 」タグを使って操作します。「 env 」タグはカメラ以外にもゲーム全体の操作をする場合に使いますが、それは後ほど解説します。

env タグの「 camerax 」「 cameray 」属性でカメラの位置をずらします。デフォルトではどちらも 0 になっています。カメラを右に移動させると、それに映されている画面全体は左にずれ、カメラを上に移動させるとそれに映されている画面全体は下にずれます。「 camerazoom 」属性を使うとカメラの拡大率が指定できます。デフォルトは 100 で、camerazoom=200 であれば画面全体が 2 倍に拡大されます。「 camerarotate 」属性を使うとカメラを回転させることができます。指定する値の意味は「 rotate 」属性と同じです。

```
*start| スタート
@ 街 昼
@ しおり 前 右 制服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@env camerax=-150 time=2000
カメラ位置を左に 150 ずらしています。

@env cameray=150 time=2000
カメラ位置を上に 150 ずらしています。

@env camerazoom=200 time=2000
カメラの拡大率を 200% にしています。

@env camerarotate=180 time=2000
カメラを反時計回りに 180 度傾けています。
```

time 属性によるアクションも使えます。アクションさせた方がカメラの動きがわかりやすいかもしれません。

カメラは単純に画面全体を動かすのにも使えますが、画像の Z 位置を指定すると 2.5 次元的な動きが実現できます。 2.5 次元というのは、画像を奥行き方向に配置できるという意味で使っています。これまでの横方向 (X) 、縦方向 (Y) に加えて、画像には奥行き方向 (Z) が存在します。デフォルトでは 100 で、数字が大きいほど手前側となります。 3 次元 (3D) ではないのでカメラが回りこんだりする事はできませんが、いくらか 3 次元っぽく 2 次元の画像が動かせます。

```
// ■ levelz
// 立ち絵の Z 位置を表示レベルごとに指定します。
levelz[0] = 100;
levelz[1] = 100;
levelz[2] = 100;

// ■ bglevelz
// 背景の Z 位置を指定します。
bglevelz = 100;
```

背景と立ち絵の Z 位置はあらかじめ定義しておく必要があります。「 init 」フォルダの envinit.otjs を開いて上のような部分を探してください。立ち絵の Z 位置は表示レ

ベルごとに「 levelz[表示レベル] = Z 位置の形で指定します。背景の Z 位置は bglevelz = Z 位置; の形で、すべての背景共通に指定します。 Z 位置は 0 より大きい数字である必要があります。ここでは以下のようにしてください。

```
// ■ levelz
//  立ち絵の Z 位置を表示レベルごとに指定します。
levelz[0] = 100;
levelz[1] = 150;
levelz[2] = 100;

// ■ bglevelz
//  背景の Z 位置を指定します。
bglevelz = 50;
```

表示レベル 2 はサンプルスクリプトでは使用していないので 100 のままです。表示レベル 1 のときの立ち絵は、表示レベル 0 の画像を 1.5 倍に拡大した画像なので、表示レベル 1 のときの Z 位置も表示レベル 0 の時の位置の 1.5 倍手前にしています。本当は距離と大きさの関係はこれほど単純ではないです。が面倒なのでこうしておきます。

背景はテキトーですが表示レベル 1 の立ち絵よりも 2 倍遠くにしています。カメラを使用する場合は、背景を準備する際にカメラからの距離も考える必要があります。場合によっては距離ごとにパーツ分けなどが必要かもしれません。カメラをちゃんと使おうとすると大変です。背景については、同人でフリー素材を使っているとアイレベルもバラバラだったり立ち絵が浮遊していたりするので、細かいことは気にしないというのもアリかもしれません。

envinit.otjs を上の通り変更して保存したら、 tools フォルダの autosetting.bat を実行して変更を適用してください。

Z 位置は直接的にカメラとの距離を指定するのではなく、画像間の相対的な位置として指定します。 Z 位置のデフォルトは 100 で、この値を基準とします。 Z 位置が 100 のとき、カメラを左に 10 移動すると画像は右に 10 ピクセル移動します。 Z 位置が 150 であれば、右に 15 ピクセル移動し、 50 であれば右に 5 ピクセル移動します。

車に乗って外を見ると手前の景色が奥の景色よりも早く流れていきます。手前のものが奥の物よりも大きく移動しているように見える、ということです。カメラが移動するというのは見ている側の車の位置が移動するのと同じです。 xpos 、 ypos 属性で画像を移動するというのは、車は停止して、車の外の人 (見られている側＝被写体)

が歩いて移動しているのと同じです。

xpos 、 ypos 属性による位置指定も影響を受けます。おなじxpos=100でも Z 位置が 100 の時は右に 100 ピクセル、 Z 位置が 150 のときは右に 150 ピクセルずれるようになります。

```
*start| スタート
@ 街 昼
@ かえで 奥 左 私服 ポーズ a 無表情
@ しおり 前 左中 制服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@env camerax=-200 time=2000
カメラ位置を左に 200 ずらしています。

@env cameray=200 time=2000
カメラ位置を上に 200 ずらしています。

@env camerazoom=200 time=2000
カメラの拡大率を 200% にしています。

@env camerarotate=180 time=2000
カメラを反時計回りに 180 度傾けています。
```

上のスクリプトを実行すると Z 位置の効果がわかると思います。

先ほど定義したとおり、背景の「街」の Z 位置は 50 、表示レベル 0(奥) の「かえで」の Z 位置は 100 、表示レベル 1(前) の「しおり」の Z 位置は 150 となってなので、カメラが左に 150 ずらしたとき、「街」は 100 ピクセル、「かえで」は 200 ピクセル、「しおり」は 300 ピクセルずつ右にずれます。結果としてどのように見えるかは実際に実行してみてください。

camerazoom 属性による拡大と camerarotate 属性による回転は Z 位置の影響を受けません。先ほどの車の例で言えば、車のなかで 5 倍双眼鏡を使うと、距離に関係なく全てのものが元の大きさの 5 倍に拡大して見えます。首を 45 度傾けても距離に関係なくすべてのものが 45 度傾きます。

```
*start| スタート
@ 街 昼
@ かえで 奥 左 私服 ポーズ a 無表情
@ しおり 前 左中 制服 ポーズ a 笑顔
@layer name= 星１ file= 星 levelz=80 xpos=-50 ypos=300
背景、立ち絵、星を表示しました。


@env camerax=-200 time=2000
カメラ位置を左に 200 ずらしています。

@env cameray=200 time=2000
カメラ位置を上に 200 ずらしています。
```

環境レイヤの Z 位置は「 levelz 」属性によって指定できます。 envinit.otjs で指定した 100 を基準とする位置指定と同じものです。

```
*start| スタート
@ 街 昼
@ かえで 奥 左 私服 ポーズ a 無表情
@ しおり 前 左中 制服 ポーズ a 笑顔
@layer name= 星１ file= 星 xpos=-270 ypos=250 zoom=80 nocamera
背景と立ち絵を表示しました。

@env camerax=-200 cameray=200 time=2000
カメラ位置を左上に 200 ずらしています。

@env camerazoom=200 camerarotate=180 time=2000
カメラの拡大率を 200% 、 180 度回転しています。
```

char 、 stage 、 layer タグでは、「 nocamera 」属性を使うことができます。この属性に true が指定された画像は camerax 、 cameray 、 camerarotate 、 camerazoom 属の影響を受けなくなります。上のスクリプトで「星１」の位置は全く動きません。画面上に日付や何かのパラメータ画像を表示する際などに使えます。

イベント絵については初めからカメラの影響を一切受けません。なので levelz 属性などで Z 位置を指定することもできません。

```
*start| スタート
@ 街 昼
@ かえで 奥 左 私服 ポーズ a 無表情
@ しおり 前 左中 制服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@env shiftx=-100 time=1000
カメラを左に 100 シフトしています。

@env shifty=100 time=1000
カメラを上に 100 シフトしています。
```

「 shiftx 」「 shifty 」属性は camerax 、 cameray 属性とほぼ同じですが、画像の Z 置とは無関係に、指定したピクセル数だけ全体が移動します。

camerax=-100では画像の Z 位置によって実際にずらされるピクセル数が変わりますが、 shiftx=-100ではすべての画像が右に 100 ピクセルずれます。shifty=100では全ての画像が下に 100 ピクセルずれます。その他の点は camerax 、 cameray 属性と同じです

```
*start| スタート
@ 街 昼
@ かえで 奥 左 私服 ポーズ a 無表情
@ しおり 前 左中 制服 ポーズ a 笑顔
@layer name= 星１ file= 星 xpos=-270 ypos=250 zoom=80 noshift
背景、立ち絵、星を表示しました。

@env shiftx=-100 time=1000
カメラを左に 100 シフトしています。

@env shifty=100 time=1000
カメラを上に 100 シフトしています。
```

nocamera 属性と似ていますが、「 noshift 」属性もあります。この属性に true を指定すると、その画像は shiftx 、 shifty 属性の影響を受けなくなります。上のスクリプトでは「星１」は全く動きません。使い所は nocamera 属性とそう変わりません。

イベント絵は、 shiftx 、 shifty 属性についても影響を受けないようになっています。

```
*start| スタート
@ 街 昼 blur=30
@ かえで 奥 左 私服 ポーズ a 無表情 blur=15
@ しおり 前 中 制服 ポーズ a 笑顔
@layer name= 星１ file= 星 xpos=-270 ypos=250 zoom=80 noshift blur=20
背景、立ち絵、星を表示しました。

@ 街 blur=15 time=1000
@ かえで blur=0 time=1000
@ 星１ blur=5 time=1000
@ しおり blur=-15 time=1000
かえでにピントを合わせています。

@ 街 blur=10 time=1000
@ かえで blur=-5 time=1000
@ 星１ blur=0 time=1000
@ しおり blur=-20 time=1000
星にピントを合わせています。
```

KAGEX のカメラには、ここまで紹介してきたような簡易的な Z 位置による XY 位置の補正の機能しかありません。使うのに苦労する割に機能としては微妙かもしれません。被写界深度やピンぼけなどを表現する機能はありません。上のスクリプトのように blur 属性でぼかすだけでもそれっぽくは見えます。

■画面揺らし

```
*start| スタート
@ 街 昼
@ しおり 前 右 制服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@quake time=1000 vmax=20 hmax=10
画面を１秒間揺らしています。

@quake time=2000 sync
画面を 2 秒間揺らしました。
```

「 quake 」タグを使うと画面全体を揺らすことができます。アクションなどでも揺らせますが quake タグはお手軽に使えて便利だったりします。「 vmax 」属性は縦、「 hmax 」属性は横に、最大で何ピクセルだけ揺らすかを指定します。これらの属性に指定した範囲内でランダムに画面が揺らされます。

vmax 、 hmax 属性は省略するとデフォルト値の 10 になります。「 time 」属性には何ミリ秒の間画面を揺らすかを指定します。デフォルトでは非同期動作ですが、「 sync 」属性で同期動作にすることもできます。

```
*start| スタート
@ 街 昼
@ しおり 前 右 制服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@quake
画面を揺らしています。

quake タグの揺れはページ送りでキャンセルされません。

@stopquake
画面揺らしを停止しました。
```

quake タグの time 属性を省略すると、いつまでも画面が揺れ続けることになります。また、 quake タグの非同期動作はアクションやトランジションなどと異なり、ページ送りでキャンセルされません。揺れを停止するには「 stopquake 」タグを使う必要があります。

```
*start| スタート
@ 街 昼
@ しおり 前 右 制服 ポーズ a 笑顔
背景と立ち絵を表示しました。

@quake message
画面を揺らしています。

ゆれゆれゆれ。

@stopquake
揺れを停止しました。
```

「 message 」属性を使うと、メッセージ部分も画面と一緒に揺らすことができます。読みづらくなるのでテキスト表示と一緒に使用するのはお勧めしません。

■拡張トランジション

トランジションの種類として「 crossfade 」「 universal 」「 scroll 」の３つを紹介し

したが、吉里吉里プラグインを使用するとその以外のトランジションが使用できます。

吉里吉里プラグインとは、吉里吉里の機能を拡張するためのプログラムで、開発用フォルダの「 plugin 」フォルダに入っている拡張子が dll のファイルがそれです。そこに入っているプラグインは KAGEX の動作に必要なのでゲームが完成した暁にはゲーム本体と共にそれらも配布することになります。プラグインを使うにはプラグインを吉里吉里に「リンク」するのですが、それら必須のプラグインのリンクは自動的に行われています。拡張トランジションの機能を含むプラグインは必須ではないためそこには含まれていません。

まずはプラグインファイルをコピーしてきます。 SDK フォルダの「プラグイン」フォルダの「 extrans 」フォルダに入っている「 extrans.dll 」を、開発用フォルダの「 plugin フォルダにコピーしてください。

```
// 追加のプラグインのリンク
// 例 : Plugins.link("saveStruct.dll");
```

次に吉里吉里にリンクします。プロジェクトフォルダの「 init 」フォルダの「 plugins.tjs 」を開くと、上のように書かれています。プラグインのリンクは「Plugins.link(" プラグインの名前 ");」のような形式になります。

```
// 追加のプラグインのリンク
// 例 : Plugins.link("saveStruct.dll");
Plugins.link("extrans.dll");
```

Plugins.link("extrans.dll");という行を追加したら保存してください。「 extrans.dll 」が吉里吉里起動時にリンクされるようになります。これで準備は整ったので拡張プラグインを使用してみましょう。

拡張プラグインは KAGEX には直接関係ないので簡単に紹介するうだけに留めておきます。詳しくは吉里吉里リファレンス (http://devdoc.kikyou.info/tvp/docs/kr2doc/contents/) のトランジションのページ (http://devdoc.kikyou.info/tvp/docs/kr2doc/contents/Transition.html#id295) を参照してください。「拡張トランジションプラグイン」を読みながら属性値を適当にいじっていればわかってきます。

```
*start| スタート
@ 街 昼
背景を表示しました。

@ 街 夕　trans=wave time=10000 wavetype=2 maxh=50 maxomega=0.2
波トランジションをしました。

@ 街 夜　trans=mosaic time=2000 maxsize=20
モザイクトランジションです。

@ 向日葵 昼　trans=turn time=2000
ターントランジションです。

@ 向日葵 夕　trans=rotatezoom time=2000 factor=0 accel=0 twist=3 twistaccel=0
回転ズームトランジションです。

@ 街 昼　trans=rotatevanish time=2000 accel=2 twist=-2 twistaccel=0
回転消滅トランジションです。

@ 街 夕　trans=rotateswap time=2000 twist=2
回転入れ替えトランジションです。

@ 街 夜　trans=ripple time=2000 rwidth=64 roundness=1.0 speed=6.0 maxdrift=20
波紋トランジションです。
```

trans 属性にトランジションの種類、 time 属性にトランジションの時間、それに加えて種類ごとの属性を指定します。詳細はリファレンスを参照してください。リファレンスにはありませんが sync 、 nosync 属性も使えます。

■トランジションの定義

よく使うトランジションをあらかじめ定義しておくことができます。開発用フォルダの tools フォルダの「 register_trans.exe 」を実行して、プロジェクトフォルダを選択してください。

「追加」ボタンをクリックすると新しくトランジションを定義できます。追加された「 newtrans 」を選択して、追加ボタンの下のテキストボックスに名前を入力、エンターキーでトランジションの名前を変更できます。ここでは「右スクロール」という名前にしておきます。「削除」ボタンをクリックで追加したトランジションを削除

することもできます。

左側のリストからトランジションを選択し、右側でそのトランジションの属性を設定します。「右スクロール」の属性は「トランジションの種類」を「 Scroll 」、「 time 」を「 1000 」、「 from 」を「 left 」、「 stay 」を「 nostay 」としてください。

それができたら、メニューの「出力」をクリックします。これを忘れると変更が反映されません。ここまで済めばスクリプト中で使えるようになります。

```
*start| スタート
@ 街 昼
背景を表示しました。

@ 向日葵 trans= 右スクロール
右スクロールで背景を変更しました。

@ 街 夕 trans= 右スクロール time=3000
time 属性を上書きして右スクロールしました。

@ 街 夜 右スクロール stay=stayfore
stay 属性を上書きして右スクロールしました。
```

trans 属性に定義したトランジション名を指定すると、そのトランジションが使用できます。それと同時にトランジションの属性を指定することで、定義された内容の一部分を変更してトランジションできます。 @ 街 夕 trans= 右スクロール time=3000 では、time 属性のみ変更され、定義されている 1 秒ではなく 3 秒でトランジションが行われるようになります。

@ 街 夜 右スクロール のように、 trans 属性の属性名は省略して、定義した名前のみで使うこともできます。基本的にこちらを使うことになるとおもいます。

■自動トランジションの定義

定義したトランジションを、画像変更時に自動的に使用するように定義しておくことができます。自動的に実行されるトランジションを「自動トランジション」といいます。

自動トランジションを定義する前に、使用するトランジションを定義しておきます。

先ほど使用した register_trans.exe をもう 1 度起動してください。「背景トランジ

ション」、「立ち絵トランジション」という 2 つのトランジションを追加しいて、トランジションの種類はどちらも「 Crossfade 」、「 time 」属性は背景トランジションが 1000 、立ち絵トランジションが 500 となるように設定します。立ち絵トランジションの方は「 nosync 」属性のチェックも入れてください。非同期トランジションになります。

以上が設定できたら「出力」メニューをクリックするのを忘れないでください。

次に自動トランジションを定義します。

```
// ■ envTrans
//   舞台、時間、イベント絵のいずれかが変更された際の自動トランジション
//   を指定します。
autoTrans["envTrans"] = "";

// ■ stageTrans
//   舞台または時間が変更された際の自動トランジションを指定します。
// envTrans よりも優先されます。
autoTrans["stageTrans"] = "";

// ■ timeTrans
//   時間が変更された際の自動トランジションを指定します。
// envTrans 、 stageTrans よりも優先されます。
autoTrans["timeTrans"] = "";

// ■ eventTrans
//   イベント絵が変更された際の自動トランジションを指定します。
// envTrans よりも優先されます。
autoTrans["eventTrans"] = "";

// ■ charTrans
//   立ち絵のポーズ、服装、表情のいずれかが変更された際の自動トランジ
//   ションを指定します。
autoTrans["charTrans"] = "";

// ■ poseTrans
//   立ち絵のポーズが変更された際の自動トランジションを指定します。
// charTrans よりも優先されます。
```

```
autoTrans["poseTrans"] = "";

// ■ dressTrans
//   立ち絵の服装が変更された際の自動トランジションを指定します。
// charTrans よりも優先されます。
autoTrans["dressTrans"] = "";

// ■ faceTrance
//   立ち絵の表情が変更された際の自動トランジションを指定します。
// charTrans よりも優先されます。
autoTrans["faceTrans"] = "";

// ■ positionTrans
//   立ち絵の x/ypos 、表示状態、表示レベルのいずれかが変更された際の自動
//   トランジションを指定します。
autoTrans["positionTrans"] = "";

// ■ charDispTrans
//   立ち絵が表示または非表示にされた際の自動トランジションを指定します。
// positionTrans よりも優先されます。
autoTrans["charDispTrans"] = "";
```

init フォルダの envinit.otjs を開いて上のような部分を探してください。autoTrans["自動トランジションの種類"] = "トランジション名"の形で定義します。「 envTrans 」など自動トランジションの種類ごとに、それぞれ使われるタイミングが異なります。 envTrans に指定したトランジションは舞台、時間、イベント絵のいずれかが変更された際に使われます。その他の自動トランジションは上のスクリプト中の説明を読んでください。

同じ変更に対して複数の自動トランジションが定義できる場合、適用範囲の狭い自動トランジションが優先されます。たとえば、時間が変更された際の自動トランジションは「 timeTrans 」「 stageTrans 」「 envTrans 」の 3 つに定義できますが、 timeTrans は時間のみ、 stageTrans は時間と舞台、 envTrans は時間と舞台とイベント絵に適用されるので、この順番に優先されます。 timeTrans が指定されていれば timeTrans 、 timeTrans が指定されていなければ stageTrans 、 stageTrans も指定されていなければ envTrans 、が時間が変更された場合の自動トランジションとして使われることになります。 envTrans も指定されていなければ自動トランジションは行われません。

```
// ■ envTrans
//   舞台、時間、イベント絵のいずれかが変更された際の自動トランジション
//   を指定します。
autoTrans["envTrans"] = "  背景トランジション   ";

// ■ charTrans
//   立ち絵のポーズ、服装、表情のいずれかが変更された際の自動トランジ
//   ションを指定します。
autoTrans["charTrans"] = "  立ち絵トランジション   ";

// ■ positionTrans
//   立ち絵の   x/ypos   、表示状態、表示レベルのいずれかが変更された際の自動
//   トランジションを指定します。
autoTrans["positionTrans"] = "  立ち絵トランジション   ";
```

長いので該当部分のみですが、サンプルとして上の 3 つを設定しておきます。これ以外は変更せず未指定のままです。指定ができたら autosetting.bat を実行しておきます。

```
*start|  スタート
@  街 昼
;  背景トランジションが実行されます
背景を表示しました。

@  しおり 前 中 制服 ポーズ   a   笑顔
;  立ち絵トランジションが実行されます
立ち絵を表示しています。

@  しおり 私服
;  立ち絵トランジションが実行されます
立ち絵の服装を変更しています。

@  しおり 消
;  立ち絵トランジションが実行されます
立ち絵を消去しています。

@  街 夕
;  背景トランジションが実行されます
時間を変更しました。
```

前のページのスクリプトを実行すると、自動トランジションしながら背景や立ち絵が表示されます。背景または時間の変更では envTrans に指定した背景トランジションが実行されます。立ち絵の表示 / 非表示や服装変更では positionTrans 、 charTrans に指定した立ち絵トランジションが実行されます。タグごとのトランジションの指定が省略できるのでかなり便利です。

```
*start| スタート
@ 街 昼 time=2000
背景を表示しました。

@ しおり 前 中 制服 ポーズ a 笑顔 sync
立ち絵を表示しました。

@ しおり 私服 time=2000
立ち絵の服装を変更しています。

@ しおり 消 sync
立ち絵を消去しました。

@ 街 夕 nosync
時間を変更しています。
```

trans 属性で定義済みトランジションを指定した場合と同様に、自動トランジションが行われるタグで一部の属性を指定して変更することができます。 @ 街 昼 time=2000 では、自動トランジションである背景トランジションの time 属性を変更し、 2 秒間の非同期クロスフェードトランジションが行われます。 @ しおり 消 sync では立ち絵トランジションの sync 属性が変更され、 0.5 秒の同期クロスフェードトランジションが行われます。

```
*start| スタート
@ 街 昼 trans=scroll from=top stay=nostay time=1000
背景を表示しました。

@ しおり 前 中 制服 ポーズ a 笑顔 notrans
立ち絵をトランジション無しで表示しました。
```

@ 街 昼 trans=scroll from=top stay=nostay time=1000 のように trans 属性を指定してしまえば自動トランジションは使われません。 trans=crossafade の省略である fade 属性でも OK です。自動トランジションと関係なく個別に好きなトランジションが実行できます。

また、 @ しおり 前 中 制服 ポーズ a 笑顔 notrans のように「 notrans 」属性を使用すると、自動トランジションが設定されていてもトランジション無しで表示させることができます。

■フェード時間の定義

```
// ■ fadeValue
//   フェードのデフォルトの時間を指定します。
fadeValue = 500;

// ■ charFadeValue
//   立ち絵のフェードのデフォルトの時間を指定します。
charFadeValue = 500;
```

それぞれミリ秒単位で指定します。「 charFadeValue 」は立ち絵、「 fadeValue 」はその以外の画像の場合のデフォルト値を指定します。 500 ミリ秒から変更する場合は書き換えて「 autosetting.bat 」を実行してください。

```
*start|  スタート
@ 街 昼  fade
背景をフェード表示しました。

@ しおり 前 中 制服 ポーズ  a  笑顔  fade
立ち絵をフェード表示しました。
```

fade 属性の属性値を省略すると、先ほど指定した時間でのクロスフェードになります。 @ 街 昼 fade では fadeValue @ しおり 前 中 制服 ポーズ a 笑顔 fade では charFadeValue に指定した時間になります。

■バックアップ

パソコンのデータはハードディスクに保存されていますが、ハードディスクは消耗品です。ゲーム開発中は必ずバックアップをとりましょう。データが消えてバックアップも無ければ全部最初からやり直しになってしまいます。

バックアップ先としては外付けハードディスクや DVD 、 USB 、オンラインストレージなどがあります。何でもいいので好きなものを使ってください。ここで重要なのはバックアップを自動化するということです。万が一に備えて毎日毎日手動で頑張れるのはよっぽど真面目な人だけです。データが消えるのは面倒臭がってしばらくさぼっているときです。最初から、とはなりませんが何週間も前からやり直しになります。自動化してしまえばそんな悲しみとも無縁でいられます。

実際のやり方は各自調べて何とかしてください。後でいいや、と思ったらアウトです。やりません。消えてからやり始めることになります。「賢者は歴史に学び、愚者は経験に学ぶ」という言葉があります。私は愚者でした。

自分は使ったことがないのですが、最近の外付けハードディスクには自動的にバックアップしてくれるソフトウェアも付属しているようです。 DropBox や SugarSync などのオンラインストレージでは、ただ保存しておけば自動的にオンライン上にバックアップを取ってくれます。

■データのやり取り

複数人で協力してゲームを製作していると、ファイルが大量発生して大変なことになります。直接会って同じ場所で作っていたら別かもしれませんが、メールやスカイプ越しでファイルをやり取りしていると凄いことになります。何度送り間違えたかわかりません。そんなことしねーよ、と思うかもしれませんがするのです。

メール越しで毎回ゲームのデータ全部を送るわけにもいかないので、前回送った分から変更されているファイルだけ送るようにしたとします。変更したデータ全てを正確に送り、全てを正確に上書きしてくれればいいのですが、そんなことは無理です。どこかで 1 度でも間違えばどんどんおかしくなっていきます。 2 人だけならまだいいですが 3 人 4 人 5 人と人間が集まれば誰かがミスります。

複数人での製作なら最低でも DropBox くらいの機能は導入してください。全員のデータの状態を同じにでき、バックアップも取れてファイルの変更履歴も取れます。お手軽で便利なのですが容量に比して少し値段が高めかなあとはおもいます。 50GB で足りるのであれば月 10 ドルで良い感じです。

■バージョン管理

バージョン管理システムというものがあります。 Subversion 、 Git 、 Mercurial 、 Bazzar など。デジタルノベル製作で実際に使われているのは Subversion になると思います。バージョン管理とは、ファイルの変更の記録を管理することです。

プログラミングをする際には特になのですが、あるファイルを昔の状態に戻したいということがあります。これはバージョン管理システムを使っていれば簡単にできます。吉里吉里のスクリプトを書いていても、書き換えてみたら動かなくなったけど直し方がわからない、なんてことがあったりします。変更前のファイルに戻してしまえば一発で解決です。

また、同じ「 01.ks 」を 2 人で同時に編集していたら困ったことになるのはわかる

とおもいます。この状態を競合といいますが、このような際にもお知らせしてくれたりします。 DropBox などのオンラインストレージでは気づかずにそのまま保存されてしまうかもしれません。

同人ゲーム製作では、過去のバージョンを残しておく用途にも使えます。夏コミ版、サンクリ版、冬コミ版、と微妙に違ったバージョンをどうやって残しておきますか? バージョン管理のタグによって簡単に残しておけます。

Subversion ですと TortoiseSVN で比較的簡単にバージョン管理システムが使えます。リポジトリを置くためのサーバーが必要ですが、 1 人であれば DropBox 上に置いてしまっても動きます。チーム製作ならサーバを借りられればベストですが、Subversion のホスティングサービスなんてのもあるようです。いずれにせよ、バージョン管理って何? という人は 1 度調べてみるといいです。

5. サウンド操作スクリプト

単純に音を鳴らすだけでなく、フェードやボリューム調整など他の機能も紹介します。キャラクターのボイスについてもこの章で扱います。

■同期 / 非同期再生

```
*start| スタート
@se 雨 nosync
効果音を非同期再生しています。

@se 雨 stop
効果音を停止しました。

@se 雨 sync
効果音を同期再生しました。

@bgm 湖畔の村 loop=false sync
BGM を同期再生しました。
```

音の再生についても、トランジションやアクションと同じく同期 / 非同期があります。同期再生では再生が終了するまでスクリプトの実行が一時停止されます。非同期再生では一時停止しません。

デフォルトでは非同期再生となっていますが、再生を開始するときに sync 属性をつけると同期再生できます。ただし、 loop 属性によってループ再生をする場合は、再生が終わることがないので必ず非同期再生になります。 BGM についても loop=false であれば同期再生できますが、あまり使うことは無いと思います。

トランジションやアクションの非同期実行と異なり、 nowait 属性をつけない非同期再生であってもページ送りで再生が停止されるようなことはありません。再生を止めるには stop 属性を使います。

```
*start| スタート
@bgm 湖畔の村 loop=false
BGM を非同期再生しています。

@bgm sync
BGM の終了を待ちました。
```

loop=falseの BGM の再生終了は@bgm syncで待つこともできます。

■フェードイン / アウト

```
*start| スタート
@bgm 湖畔の村 time=2000
BGM をフェードイン再生しています。

@se 雨 loop time=2000
効果音をフェードイン再生しています。

@bgm stop time=2000
BGM をフェードアウトしています。

@se 雨 stop time=2000
効果音をフェードアウトしています。
```

再生する際に time 属性でミリ秒単位の時間を指定すると、その時間でフェードイン再生されます。 stop 属性で停止する際も time 属性をつけるとフェードアウトできます。 BGM 、効果音で共通です。

```
*start| スタート
@bgm 湖畔の村 time=2000 sync
BGM を同期フェードイン再生しました。

@bgm stop time=2000 sync
BGM を同期フェードイアウトしました。
```

BGM のフェードイン、アウトについては sync 属性をつけると同期動作させることができます。この場合、 BGM の再生が終了するまでではなく、フェードが終了するまでスクリプトの動作が一時停止します。

```
*start| スタート
@se 雨 time=2000 sync
効果音を同期再生しました。

@se 雨 loop
効果音をループ再生しています。

@se 雨 stop time=2000 sync
効果音を同期フェードアウトしました。
```

効果音については、再生開始時のフェードインで sync 属性をつけると、フェードについての同期ではなく再生についての同期として動作します。 @se 雨 time=2000 sync

では、フェードが終了するまでではなく、再生が終了するまでスクリプトの動作が一時停止することになります。 stop 属性で停止する際は BGM と同じく、 sync 属性をつけるとフェードアウトが終了するまでスクリプトの実行が一時停止します。

■音量指定

BGM の音量は、「全体音量」と「 BGM 音量」から決まります。効果音の音量は「全体音量」と「効果音音量」から決まります。全体音量は BGM 、効果音、ボイス、ムービーの全ての音量にかかわる、文字通りゲーム全体での音量設定です。上部メニューの「音量設定」の「全体音量」からプレイヤーが設定できるようになっています。「 BGM 音量」と「効果音音量」では BGM と効果音の音量をそれぞれ調整できます。こちらも「音量設定」メニューから設定できます。

これらはプレイヤー側が設定できる値ですが、さらにスクリプト中で個別に音量調整することもできます。

```
*start| スタート
@bgm 湖畔の村 fade=80
80% の音量で BGM を再生しています。

@bgm fade=40
BGM の音量を 40% にしました。

@bgm stop
@se 雨 loop
BGM を停止して効果音を再生しています。

@se 雨 fade=50
効果音の音量を半分にしました。

@se 雨 fade=100
効果音の音量を元に戻しました。
```

「 fade 」属性で再生ごとに個別の音量を設定できます。 100 を指定するとそのままの音量、 50 を指定すると半分の音量、 0 を指定すると無音になります。 100 より大きい数字を指定することはできません。再生開始時か再生途中かに関わらず音量を変更できます。

fade 属性で指定する音量は「全体音量」「 BGM 音量」「効果音音量」とはまた別のものです。それら３つでプレイヤーによって設定された音量から、さらにどれだけ

の音量とするかを指定します。実際に BGM が再生される音量は、「全体音量」「 BGM 音量」「 fade で指定された値」が掛け算された値となります。３つ全て 100 の時の音量を 100 × 100 × 100=1,000,000 とすると、３つ全て 50 の時の音量は 50 × 50 × 50=125,000 で 1/8 となります。効果音についても BGM 音量が効果音音量となるだけで全く同じです。

スクリプト中はこんな面倒なことを考える必要はありません。「全体音量」「 BGM 音量」「効果音音量」はプレイヤーが好きに調整する値なので気にせず、 fade 属性に指定する値だけを考えればいいです。 fade=50とすればデフォルトの fade=100の時の 2 分の 1 の音量となります。fade=10なら 10 分の 1 です。

```
*start| スタート
@bgm 湖畔の村
BGM を再生しています。

@bgm fade=50 time=2000
BGM の音量を 2 秒間のフェードで半分にしました。

@bgm stop
@se 雨 loop
BGM を停止して効果音を再生しています。

@se 雨 fade=50 time=2000
効果音の音量を 2 秒間のフェードで半分にしました。

@se 雨 fade=100 time=2000
効果音の音量を 2 秒間のフェードで元に戻しました。
```

「 fade 」属性を「 time 」属性を一緒に使うことで、いきなり音量を変更せずにフェードしながら音量を変更できます。

```
*start| スタート
@bgm 湖畔の村 fade=50 time=2000 sync
半分の音量で BGM を再生開始しました。

@bgm fade=100 time=2000 sync
BGM の音量を 2 秒間の同期フェードで元に戻しました。

@bgm stop
@se 雨 fade=50 time=2000 sync
; 効果音再生時の「 sync 」なので再生終了までの同期動作
BGM を停止して効果音を再生しました。

@se 雨 loop
効果音を再生開始しました。

@se 雨 fade=50 time=2000 sync
; 効果音再生途中の「 sync 」なのでフェード終了までの同期動作
効果音の音量を 2 秒間の同期フェードで半分にしました。
```

fade 属性と time 属性によるフェードも sync 属性をつけることで同期動作にできます。このときの動作はフェードイン、アウトと同じです。つまり、 BGM については sync 属性をつければ音量フェードが終わるまでスクリプトの実行が一時停止します。効果音については、再生開始時に sync 属性をつけると効果音再生終了までの同期動作となります。再生途中でのフェードで sync 属性をつけるとフェード終了までの同期動作となります。

１つ注意として、 fade 属性の値は次の再生開始時に fade 属性が省略された場合 100 に戻されます。@se 雨 loop では、直前に @se 雨 fade=50 time=2000 sync で fade 属性に 50 が指定されているため半分の音量で再生されるように思うかもしれません。しかし一度再生は終了しているので fade の音量は 100 に戻されて再生します。ここでも 50 で再生したければ@se 雨 loop fade=50 のように再び fade 属性を指定する必要があります。ずっと前に指定した fade 属性の値が残っていたりするとバグのもとになるので、再生開始ごとに 100 に戻されるようになっています。これは BGM についても同じです。

■音量設定について

前提として、 BGM や効果音は WAV ファイルの段階でバランスよく音量を調整しておきます。 fade 属性はファイルごとの音量のバランスを整えるための機能ではありません。「全体音量」「 BGM 音量」「 SE 音量」の初期値 (ゲームを初めて実行した時設定されている値) も開発ツールで設定できるので、そちらも良い感じに設定しておきます。せっかくファイル間の音量バランスは良くしても、ゲーム全体で音量が大きすぎれば起動時に爆音が流れてびっくりします。逆に小さすぎて聞こえないというのも問題です。「全体音量」「 BGM 音量」「効果音音量」が全て 100 のときにちょうど良くなるようにファイル自体の音量を調整してしまうことが多いです。本当は３つ全ての初期値を「 50 」にして、それで音量がちょうど良くなるように WAV ファイルの音量を調整するのがベターです。こうするとプレイヤーが音量を大きくも小さくもできます。初期設定が 100 では小さくすることしかできないので少し不便かもしれません。

fade 属性の使いどころは、ゲーム中で動的に音量を変更しなければならない場合です。「動的」というのは状況に応じて動作などが変わることです。対義語は「静的」となりますが、こちらはどのような状況でも動作などが変わらないことです。

ゲームについて言えば、プレイヤーの動作で変わっていくゲームの状態は動的な要素となります。変わらない物は静的な要素となります。アクションゲームの敵キャラクタはプレイヤーによって倒されたりされなかったりする動的な要素です。ゲーム内の地形などは変化しない静的な要素となります。最近は地形も破壊できたりと動的な要素であることもありますがそれはおいておきます。動的な要素の存在は他のメディアと比較してのゲームの特徴です。映画は完全に静的なメディアです。観客が涙を流そうが爆睡しようが映画の内容は寸毫も変わりません。

デジタルノベルは他のジャンルのゲームと比較して動的な要素が少ないため「静的ゲーム」と呼ばれることもあります。アクションゲームなどは「動的ゲーム」です。ただ、デジタルノベルにも動的な要素は存在します。典型的には選択肢によって変化すればそれは動的な要素といえます。

```
*start| スタート
@se 雨 loop
音量を選択して下さい

@seladd text=50% target=*50 パーセント
@seladd text=100% target=*100 パーセント
@select
```

```
*50 パーセント
@se 雨 fade=50
音量を 50% にしました。

@next target=*start
*100 パーセント
@se 雨 fade=100
音量を 100% にしました。

@next target=*start
```

上のスクリプトではプレイヤーの選択によって効果音の音量を変更しています。このような場合は音量が異なるファイルを２つ準備するよりも fade 属性を使ったほうが便利だと言えます。直接選択肢によらずとも、キャラクタの好感度やゲームの進行度によって音量が変わる、ということはあり得るかもしれません。

次は選択肢ではなく、プレイヤーの単純なクリックによる動的な動作の例です。

```
*start| スタート
@bgm 湖畔の村
BGM を再生しています。

再生中です。

@bgm fade=50
音量を半分にしました。
```

再生中です。の後にクリックすると次に進み音量が変更されますが、プレイヤーのクリックタイミングによって曲のどの部分から音量を変更するのかが動的に変わってきます。曲の先頭から 5 秒後からかもしれませんし 10 秒後からかもしれません。その全ての場合の音楽ファイルを準備することはできないので fade 属性を使って音量を変化させます。

fade 属性は「 @se 雨を使う際は常に fade=50 にする」というような使い方をするものではありません。それは wav ファイルの段階で調整してください。 fade 属性はゲーム中で変化させる必要がある場合のみに使用します。これは次のパンについても同じです。常に右から左に移動するのであれば、そのような wav ファイルを作れば良いのです。

■パン

```
*start| スタート
@se 雨 loop
効果音を再生しています。

@se 雨 pan=100
右から聞こえるように設定しました。

@se 雨 pan=-100
左から聞こえるように設定しました。

@se 雨 pan=0
中央に戻しました。
```

効果音について、「 pan 」属性を使うとパン（ 音の聞こえてくる方向 ）を指定できます。プラスの値を指定すると右、マイナスの値を指定すると左から聞こえてくるようになります。デフォルトでは 0 で、 -100 ～ 100 の値が指定できます。 pan 属性は今のところ効果音のみで BGM では使用できません。もしかしたら今後使えるようにするかもしれません。

```
*start| スタート
@se 雨 loop
効果音を再生しています。

@se 雨 pan=100 time=2000
音を右に移動させています。

@se 雨 pan=-100 time=2000 sync
音を左に移動させました。

@se 雨 pan=0 time=2000 sync
中央に戻しました。
```

time 属性を使うこともできます。指定された方向まで指定された時間をかけて聞こえてくる方向が移動します。 sync 属性で同期動作させることもできます。注意点も fade 属性と同じです。再生開始時の sync 属性は再生終了までの同期動作となります。 pan 属性は次の再生開始時に 100 ではなく 0 に戻されます。

ここまで紹介してきた音の sync による同期動作では「 canskip 」属性が使用できます。 false を指定するとクリックで同期動作をスキップすることができなくなります。

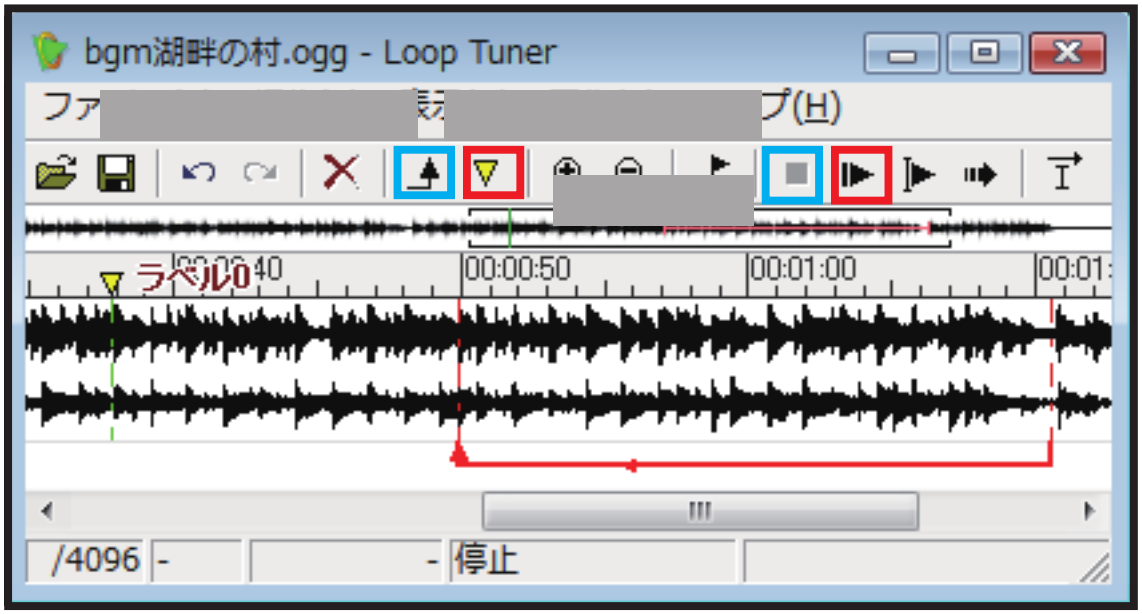
デフォルトでは　true　となっているのでスキップできます。

■ループチューナ

　吉里吉里には「ループチューナ」という開発ツールが付属しています。この使い方を少しだけ解説します。先頭から末尾のループ以外にも途中でループさせたり、音楽ファイルにラベルという印を付けることができたりします。

　tools　フォルダの「 krkrlt.exe 」がループチューナです。起動してみてください。「ファイル」メニューの「開く」から設定するファイルを選択します。ここでは例としてプロジェクトフォルダの「 bgm 」フォルダに入っているはずの「 bgm　湖畔の村　.ogg 」を開きます。

　メニューの右の方の黒い左向きの三角のボタンで現在のファイルを再生できます。黒い四角のボタンで再生停止します。



　メニュー真ん中あたりの黄色い下向き三角のボタンでラベルが新規作成できます。クリックすると下の波形の上に黄色い三角が出てきたと思います。それがラベルです。何個でも作れます。作ったラベルは、波形の上の黄色い三角をドラッグすると位置を変更できます。ダブルクリックで名前を変えることもできます。

　リンクを作るには、ラベル新規作成ボタンの左隣のボタンをおします。一番下に赤い矢印が出てきます。それがリンクです。リンクしたところでループするようになります。

　作ったラベルとリンクは「ファイル」メニューの「保存」で保存できます。保存す

ると「 bgm 湖畔の村 .ogg 」と同じフォルダに「 bgm 湖畔の村 .ogg.sli 」というファイルができますが、それがループチューナのデータです。

詳細なループチューナの使い方は吉里吉里リファレンスの説明 (http://devdoc.kikyou.info/tvp/docs/kr2doc/contents/LoopTuner.html) を参照してください。私も勘と慣れでテキトーに使っているので説明できません。

音を作っている人などがいればその人に任せてしまうのがいいです。おそらく使いこなしてくれるでしょう。

ひとまずサンプルとして使いたいので「湖畔の村 .ogg 」はリンクを１つと「ラベル 0 」という名前のラベルを 1 つ作成したら保存してください。

音のプロフェッショナルな方はもっと良い専用ツールを使って「サンプル数」なるもので位置を指定してきます。この場合は設定ファイルを直接いじってしまいます。「 bgm 湖畔の村 .ogg 」と同じフォルダにある「 bgm 湖畔の村 .ogg.sli 」をテストエディタで開いてください。

```
Link { From=2504685;        To=1525930;        Smooth=False; Condition=no;
RefValue=0;        CondVar=0; }
```

ここでは長すぎて途中で改行されてますが、上のような行があると思います。「 From= 」の後の数字がループ元のサンプル数、「 To= 」の後の数字がループ先のサンプル数になります。ここでは From=3112941 と To=1333418 に書き換えて保存してください。きれいにループするようになります。ちゃんとループしているか、もう 1 度ループチューナで「 bgm 湖畔の森 .ogg 」を開いて再生してみると確認できます。

ループチューナの設定が保存された拡張子が sli のファイルは、元の音楽ファイルと同じフォルダに置いておけば、再生時に吉里吉里が勝手に読み込んでくれます。

```
*start| スタート
@bgm 湖畔の村
BGM を再生しています。
```

ただ BGM を再生しているだけですが、いつまでも途切れずにループチューナで設定されたリンクの位置でループされているのがわかると思います。

```
*start| スタート
@bgm 湖畔の村 start= ラベル 0
SE を再生しています。
```

「 start 」属性にループチューナで設定したラベルの名前を指定すると、そのラベルの位置から再生できます。効果音でもラベルが設定してあれば start 属性が使えます。

■ボイス再生

同人ゲームでもボイスの入っているゲームは珍しくなくなっています。 KAGEX では標準でボイス再生のための機能がついています。

サンプル素材フォルダの「 voice 」フォルダに入っている「 cv しおり _0001.ogg 」などのファイルを、全てプロジェクトフォルダの「 voice 」フォルダにコピーしてください。ボイスファイルのファイル名は、「 cv[キャラ名]_[4 桁の番号] 」という形になります。

```
*start| スタート
@ しおり voice=1
【しおり】「こんにちは」

【しおり】「ボイス再生のテストです。」

名前をつけなければボイスは再生されません。

【しおり】「ボイスの番号は自動的に１ずつ上がっていきます。」
```

ボイスを再生するには char タグを使います。 char タグは立ち絵だけでなく、キャラクタの操作全般に使用します。ここではキャラ名をタグ名にする省略記法を使っています。

「 voice 」属性にボイスの番号を指定すると、次は【しおり】で名前を表示するときにその番号のファイルが再生されます。 【しおり】「こんにちは」で「 cv しおり _0001 」が再生されることになります。そしてその次の名前表示では、その前の番号に 1 を足した番号のボイスが自動的に再生されます。 【しおり】「ボイス再生のテストです。」では「 cv し お り _0002 」【しおり】「ボイスの番号は自動的に１ずつ上がっていきます。」では「 cv し お り _0003 」が自動的に再生されます。

```
*start| スタート
@ しおり voice=1
; cv しおり _0001 が再生されます
【しおり】「こんにちは」

@ かえで voice=1
```

```
; cv かえで _0001 が再生されます
【かえで】「こんにちは」

; cv しおり _0002 が再生されます
【しおり】「ボイス再生のテストです。」

; cv かえで _0002 が再生されます
【かえで】「ボイスの番号はキャラクタごとに管理されています。」

; cv かえで _0003 が再生されます
【かえで】「他のキャラクタの番号には影響されません。」
```

ボイスの番号はキャラクタごとにそれぞれ管理されています。コメントで書いてある通りに再生されます。キャラクタが 3 人以上いても同じです。

```
*start| スタート
@ しおり voice=1
; cv しおり _0001 が再生されます。
【しおり / ？？？？】「こんにちは」
```

テキスト表示でやりましたが、半角スラッシュの後ろに実際に表示する名前を指定できます。半角スラッシュの前の名前はボイス再生の際にどのキャラかを判定するのに使わています。

```
*start| スタート
@ しおり voice=1
; cv しおり _0001 が再生されます
【しおり】「こんにちは」

@ しおり voice=5
; cv しおり _0005 が再生されます
【しおり】「番号は指定すればいつでも変えられます。」

@ しおり voice= ボイス１
; ボイス１ が再生されます
【しおり】「ファイル名で指定することもできます」

; cv しおり _0006 が再生されます
【しおり】「ファイル名が指定されたセリフでは番号が足されません。」
```

ボイスファイルの番号が連続していない部分は voice 属性にその都度番号を指定します。また、ボイスに追加があった場合など、ファイル名が特殊な場合はそのファイル名を指定して再生することもできます。

```
*start| スタート
@ しおり voice=1
; cv しおり _0001 が再生されます
【しおり】「こんにちは」

@ しおり voice=ignore
【しおり】「ここではボイスが再生されません。」

; cv しおり _0002 が再生されます
【しおり】「ボイス再生のテストです。」
```

voice 属性に「 ignore 」を指定すると、次のそのキャラの名前表示でボイスが再生されません。その次からは通常通り再生されます。

```
*start| スタート
@ しおり voice=1
; cv しおり _0001 が再生されます
【しおり】「こんにちは」

@ しおり voice=clear
【しおり】「これ以降、しおりのセリフでボイスが再生されません」

【しおり】「ここでも再生されません。」

【しおり】「ボイス無しです。」

@ しおり voice=2
; cv しおり _0002 が再生されます
【しおり】「ボイス再生のテストです。」
```

voice 属性に「 clear 」を指定すると、次以降のそのキャラの名前表示でボイスが再生されなくなります。また voice 属性で番号を指定すればボイス再生を再開できます。

```
*start| スタート
@ しおり playvoice=2 nosync
; cv しおり _0002 が再生されます
```

```
名前表示に関係なく再生できます。

@ しおり playvoice= ボイス１
; ボイス１ が再生されます
ファイル名で指定することもできます。

@ しおり playvoice=3 waitvoice
; cv しおり _0003 が同期再生されます
ボイスを同期再生しました。
```

voice 属性は次の名前表示で再生するボイスを指定する属性でしたが、「 playvoice 」属性を使うとその場ですぐにボイスファイルを再生できます。 voice 属性と同じく番号かファイル名で指定します。 playvoice 属性では sync 属性で同期再生することはできません。代わりに「 waitvoice 」属性が使えます。

音声を再生するだけなら、効果音を再生するように @se play=cv しおり _0002 でもできてしまいますが、ボイス扱いのものは voice 属性や playvoice 属性で再生するようにしてください。効果音として流すと効果音音量が適用されるため、ボイス音量が適用されなくなります。

ボイスの音量は「全体音量」「ボイス全体音量」「ボイス個別音量」で決まります。全体音量は BGM や効果音と共通です。「ボイス全体音量」は「ボイス設定」メニューから設定できます。「ボイス個別音量」は、キャラごとの音量設定です。デフォルトではメニューに表示されていませんが、ツールでキャラ名を設定すると「ボイス設定」メニューに表示させることもできます。

```
*start| スタート
@ しおり playvoice= ボイス１
ボイス再生中です。

@ しおり stopvoice
ボイスを強制的に停止しました。
```

「 stopvoice 」属性を使うと、そのキャラクタのボイスを強制的に停止することもできます。上のスクリプトで、ボイスが終わる前に ボイス再生中です。をクリックで飛ばすと、ボイスが途中でも停止します。

■遅延実行

一旦ボイスからは離れますが、「遅延実行」はボイスと共によく使われるのでここで解説します。遅延実行ではタグを実行するタイミングを遅延させることができます。

```
*start| スタート
@ しおり 前 中 制服 ポーズ b 無表情
立ち絵を表示しました。

@ しおり 左
@ しおり 笑顔 delayrun=2000
立ち絵の表情を変更します。
```

「 stage 」「 char 」「 event 」「 layer 」「 env 」「 bgm 」「 se 」タグでは「 d
が使えます。すべてのタグは上から順番に実行されていきますが、「 delayrun 」属性にミリ秒単位の時間を指定することで、その効果が現れるタイミングをずらすことができます。 @ しおり左 ではすぐに立ち絵が左に移動しますが、 @ しおり 笑顔 delayruun=2000 では 2 秒後に表情が変わることになります。 delayrun 属性がついたタグは書かれている場所では一見無視され、指定された時間が経過するまでその効果が現れません。

```
*start| スタート
@layer name= 星１ file= 星 show
星を表示しました。

@ 星１ xpos=300 time=3000
@ 星１ opacity=0 time=1000 delayrun=2000
星は２秒後から消え始めます。
```

表情変更に限らず遅延実行できます。上のスクリプトでは @ 星1 opacity=0 time=1000 というタグが delayrun 属性によって２秒後に実行されます。結果として、星は２秒後から１秒かけて透明になっていくことになります。

```
*start| スタート
@ しおり 前 中 制服 ポーズ b 無表情
立ち絵を表示しました。

@ しおり 左
@ しおり 右 delayrun=10000 sync
立ち絵を１０秒後に右に移動します。

遅延実行はページ送りでキャンセルされます。
```

nowait 属性をつけない非同期アクションや非同期トランジションはページ送りでキャンセルされましたが、遅延実行も同じくページ送りでキャンセルされます。すぐに遅延実行されるタグの実行終了後の状態になります。遅延実行で nowait 属性は使えないので必ずキャンセルされることになります。また、遅延実行を sync 属性などで同期動作にすることもできません。必ず非同期動作になります。

```
*start| スタート
@ しおり 前 中 制服 ポーズ b 無表情
立ち絵を表示しました。

@ しおり voice=10
@ しおり ポーズ a 笑顔 delayrun= ラベル 0
【しおり】「ボイスに合わせてタグを実行することができます。」
```

delayrun 属性にボイスファイルのラベル名を指定すると、その再生に合わせてタグを実行できます。「 cv しおり _0010.ogg 」をループチューナで開くとわかりますが、「タグを実行することができます。」というセリフの「タグ」の直前に「ラベル 0 」という名前のラベルが挟まれています。ボイスがその部分まで再生されたときに @ しおり ポーズ a 笑顔 実行されることとなります。ラベルを複数置けばセリフ中に何個でも遅延実行させることができます。ボイスが指定されたラベル位置まで再生される前にページ送りが発生した場合は、時間を指定した場合と同じように遅延実行はキャンセルされます。

■表情指定のスクリプトとか

遅延実行で最もよく使うのは最後に紹介したボイスの再生に合わせて表情を変更したりアクションを実行したり、という場合になると思います。便利ですが遅延実行を使用するとスクリプトの作業量が跳ね上がるので覚悟したほうがいいです。途中で力尽きると前半は動きまくるのに後半は何だかしょぼい、というゲームになりかねません。

「デバッグ」メニューの「ループチューナ」をクリックすると、最後に再生したボイスファイルをループチューナで開くことができます。「 Shift+C 」のショートカットキーでも開けます。これを活用するとボイスファイルにラベルをつけるのが楽になります。大量にあるボイスファイルからお目当てのファイルを探し出すより、それが使われる場面までゲームを進めて Shift+C の方が早いです。

実際のスクリプト作業としては立ち絵の表情変更が大部分を占めることが多いです。最初にテキストとボイスの番号の指定を済ませたスクリプトを用意して、それを実行しながらボイスのラベル付けやスクリプトの編集をしていきます。この際、編集

する部分の直前でセーブしておくと、ゲームを再起動する代わりにロードするだけで動作が確認できるので便利です。ロードの際にロード先のスクリプトを読み込みますが、これを利用して編集済みのスクリプトを読みこませることができます。編集前と編集後でラベルの位置が違っていたりするとおかしくなる場合もあるので、この時は一度ゲームを再起動してセーブしなおしになります。

「デバッグ」メニューの「立ち絵参照ウィンドウ」を使うと立ち絵の表情の組み合わせを見ながら作業できるのでもうちょっと便利になります。「立ち絵変更」という小さいウィンドウの方を閉じてしまった場合は「ファイル」メニューの「サブウィンドウ表示」から再度開けます。使い方はいじっていればなんとなくわかってくると思います。

ゲーム本体のウィンドウ、 KAGEX ログ、立ち絵参照ウィンドウ、立ち絵変更ウィンドウ、ループチューナ、テキストエディタと開いていくと、ウィンドウがどんどん増えてきます。これに加えてコンソールや画像編集ソフトやエクセルのファイル名対応表なども開いたりします。作業中は Twitter や Skype は閉じましょう。まあ、まともにやろうとするとモニタ 1 枚では足りないです。デュアルモニタおすすめです。執筆時点では自分も 2 枚ですが、お金ができたら３枚以上にする予定です。印刷できるものは紙に印刷してモニタの横に貼っておく、というのもいいと思います。

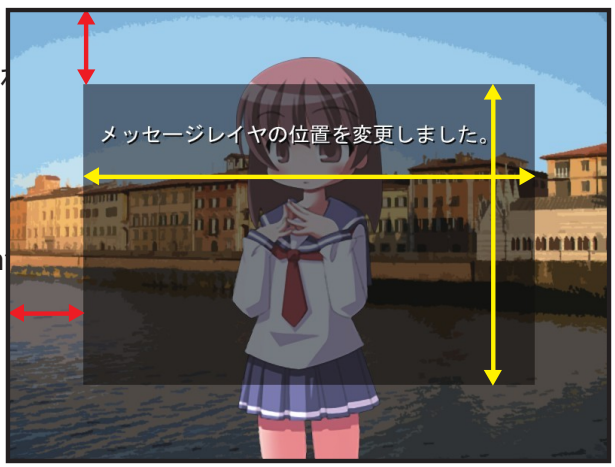
6. メッセージ操作スクリプト

テキスト表示に関連するスクリプトを紹介します。テキストの位置を変更したり、テキスト枠として画像を表示させたりします。

■テキストの表示位置

```
*start| スタート
@ 街 昼
@ しおり 前 中 制服 ポーズ b 無表情
背景と立ち絵を表示しました。

@position layer=message0 page=fore left=100 top=100 width=600 height=400
color=&RGB(0,0,0) opacity=127
; 長すぎて途中で改行されてますが 1 行に書いてください。
メッセージレイヤの位置を変更しました。
```

メッセージレイヤの位置やサイズは「 position 」タグで変更することができます。「 layer 」属性と「 page 」属性で、どのメッセージレイヤの設定を変更するのかを指定します。 layer 属性にはメッセージレイヤの番号を指定します。メッセージレイヤは 1 つだけでなく複数使用する事ができるのですが、これまで使っていたのは 0 番なので layer=message0 とします。番号の前には必ず「 message 」と付きます。 page 属性には表ページの場合「 fore 」、裏ページの場合は「 back 」を指定します。メッセージレイヤには「表ページ」と「裏ページ」というものがあります。これについては後述します。とりあえず page=fore としておきます。

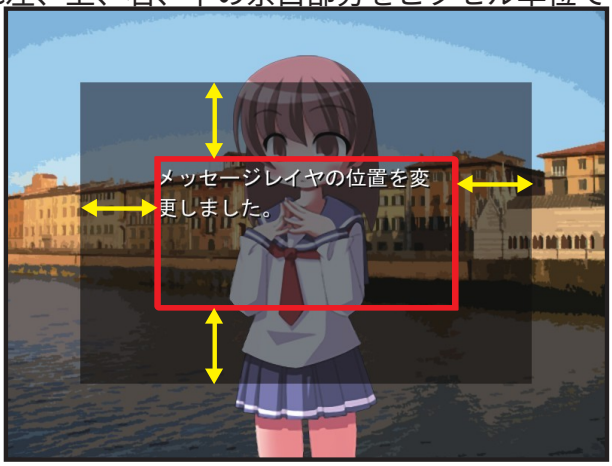


「 left 」属性と「 top 」属性に、それぞれ横、縦方向のメッセージレイヤ位置を指定します。ウィンドウ左上端からのピクセル数を指定する形になります。「 width 」属性と「 height 」属性にはメッセージレイヤのサイズを指定します。それぞれ幅、高さをピクセル単位で指定します。

「 color 」属性には color=&RGB(赤 , 緑 , 青) の形でメッセージレイヤの色を指定します。赤、緑、青のそれぞれに 0 ～ 255 で色の強さを指定します。ここでは color=&RGB(0,0,0) なので真っ黒になります。「 opacity 」属性にはレイヤの不透明度を設定します。立ち絵などで指定できる opacity 属性と同じく 0 ～ 255 を指定します。ここでは opacity=127 で半透明にしています。

```
*start| スタート
@ 街 昼
@ しおり 前 中 制服 ポーズ b 無表情
@position layer=message0 page=fore left=100 top=100 width=600 height=400
color=&RGB(0,0,0) opacity=128 marginl=100 margint=100 marginr=100
marginb=100
メッセージレイヤの位置を変更しました。
```

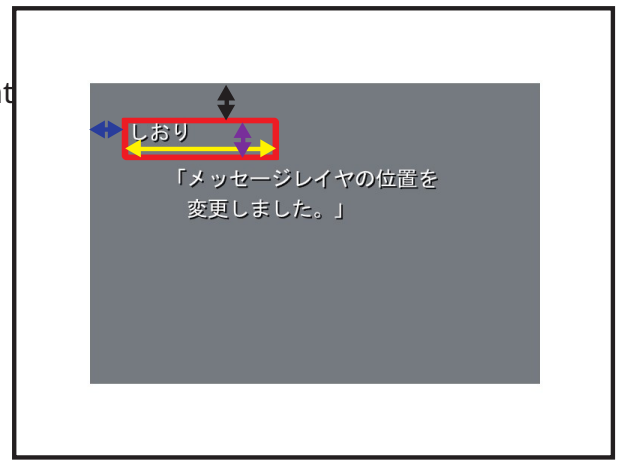
「 marginl 」「 margint 」「 marginr 」「 marginb 」属性で、メッセージレイヤの中のテキスト表示位置を指定します。それぞれ左、上、右、下の余白部分をピクセル単位で指定することで、テキストの位置を決定します。テキストはその余白の内側に表示されることになります。



上のスクリプトの場合、右の画像の赤い枠の内側にテキストが表示されることになります。テキストの右側に 1 文字分の余裕がありますが、これは禁則処理のための余裕になっています。

```
*start| スタート
@ 白
@position layer=message0 page=fore left=100 top=100 width=600 height=400
color=&RGB(0,0,0) opacity=128 marginl=100 margint=100 marginr=100
marginb=100 nameleft=50 nametop=50 namewidth=200 nameheight=50
【しおり】「メッセージレイヤの位置を変更しました。」
```

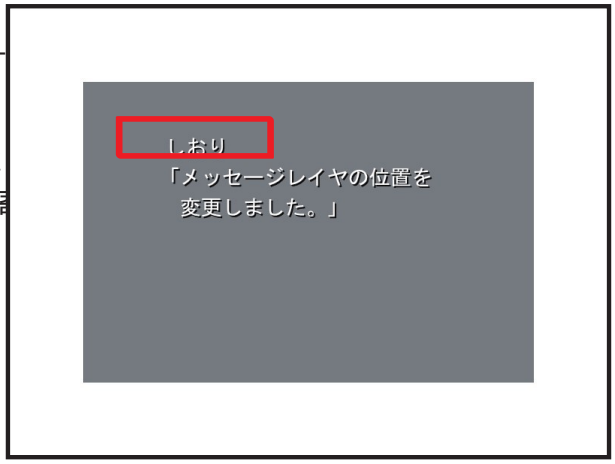
「 nameleft 」属性と「 nametop 」属性で、それぞれ横、縦方向の名前欄の位置を指定します。メッセージレイヤ左上端からのピクセル数で指定します。「 namewidth 」属性と「 nameheight 」属性には名前欄のサイズを指定します。それぞれ幅、高さをピクセル単位で指定します。



細かくて見づらいですが右の画像の赤い枠の内側に名前が表示されることになります。

```
*start| スタート
@ 白
@position layer=message0 page=fore left=100 top=100 width=600 height=400
color=&RGB(0,0,0) opacity=128 marginl=100 margint=100 marginr=100
marginb=100 nameleft=50 nametop=50 namewidth=200 nameheight=50
namealign=0 namevalign=1
【しおり】「メッセージレイヤの位置を変更しました。」
```

「 namealign 」属性と「 namevalign 」属性で、名前欄の中で名前をどこに表示するかを指定します。 namealign 属性は横方向の位置で、「 -1 」で左寄せ、「 で中央寄せ、「 1 」で右寄せになります。 namevalign 属性は縦方向の位置で、「 -1 」で上寄せ、「 0 」で中央寄「 1 」で下寄せとなります。上のスクリプトでは namealign=0 namevalign=1 で横方向は中央、縦方向は下に寄せる設定にしてあります。



■色の指定について

パソコンのモニタでは赤、緑、青の 3 色の光を組み合わせて全ての色を表現する「加法混色」 (RGB カラー) という方法が採られています。赤 (Red) 、緑 (Green) 、青 (Bl がそれぞれ 0 ～ 255 の 256 段階で、最大 256×256×256=16,777,216 色表示できます。「約 1700 万色」というやつです。

それぞれの数字が大きいほどその色の光が強くなります。 (赤 , 緑 , 青) がそれぞれ (255,0,0) なら赤、 (0,255,0) なら青、 (0,0,255) なら緑となります。 (255,255,255) で全ての光の強さを最大にすると白となります。詳しくは加法混色で調べてください。

メッセージレイヤについては、色で指定せず、代わりにテキスト枠の画像を指定してしまうことが多いので color 属性はそれほど使いません。

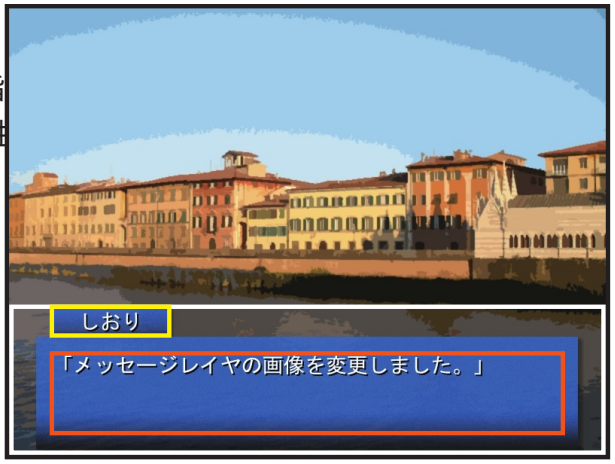
■テキスト枠

position タグの color 属性では単色の四角いメッセージレイヤしか表示できませんでしたが、画像を使用することで好きな形と色をメッセージレイヤに表示することができます。

サンプルの uiimage フォルダの「 message.png 」を表示させてみます。プロジェクトフォルダの「 uiimage 」フォルダにコピーしておいてください。

```
*start| スタート
@ 街 昼
@position layer=message0 page=fore frame=message opacity=255 left=0
top=400 marginl=60 margint=50 marginr=60 marginb=40 nameleft=60
nametop=0 namewidth=156 nameheight=37 namealign=0 namevalign=0
【しおり】「メッセージレイヤの画像を変更しました。」
```

画像を使用する場合は、 color 属性の代わりに「 frame 」属性に画像のファイル名を指定します。 message.png は元から半透明の画像なので opacity 属性は 255 で不透明にしてあります。



left 、 top 画像でレイヤの位置は指定しますが、 width 、 height 属性のサイズ指定は必要ありません。メッセージレイヤは frame 属性に指定した画像と同じサイズになります。テキスト位置や名前欄の位置についても画像を使用しない場合と同じように指定します。上のスクリプトでは右の画像のように指定しています。

■メッセージレイヤの表示 / 非表示

```
*start| スタート
@ 街 昼
背景を表示しました。

@msgoff
@msgon
メッセージレイヤを一時消去しました。

@msgoff fade=1000
@msgon trans=scroll time=1000 from=top stay=nostay
メッセージレイヤを一時消去しました。
```

「 msgoff 」タグを使用すると、メッセージレイヤを非表示にできます。非表示にしたメッセージレイヤは「 msgon 」タグで再び表示できます。

msgoff 、 msgon タグでは trans 属性や fade 属性でトランジションを指定することもできます。このトランジションは必ず同期動作になり、 nosync 属性を使うことはできません。 @msgoffのようにトランジションの指定を省略した場合は、自動トラン

ジションのところで紹介した、 fadeValue に指定した時間でのクロスフェードトランジションとなります。

```
*start| スタート
@ 街 昼
背景を表示しました。

@msgoff
メッセージレイヤを一時消去しました。
```

メッセージレイヤを一時消去しました。では、メッセージレイヤが消えた状態でテキストを表示させようとしています。このような場合は、テキストを表示する直前に自動的に @msgonが実行されてメッセージレイヤが表示されるようになっています。

最初のテキストが表示される前はメッセージレイヤが非表示になっています。なので、 背景を表示しました。の直前でも @msgonが自動的に実行されトランジションでメッセージレイヤが表示されるような動作になっています。

```
*start| スタート
@ 街 昼
背景を表示しました。

@position layer=message0 page=fore visible=false
トランジションなしでメッセージレイヤを一時消去しました。
```

@msgoff や @msgon では、自動的にクロスフェードトランジションが行われます。トランジション無しでメッセージレイヤを消去したい場合は position タグの「 visible 」属性を使用します。 true を指定すれば表示、 false を指定すれば非表示となります。

```
*start| スタート
@ 街 昼
背景を表示しました。

@ 夕 trans=crossfade time=1000 msgoff
時間を変更しました。

@ しおり 前 中 制服 ポーズ b 無表情 trans=crossafade time=500 msgoff
【しおり】「立ち絵を表示しました。」
```

トランジションの所では紹介しませんでしたが、 trans 属性によるトランジションでは「 msgoff 」という属性を一緒に使うことができます。この属性に true を指

定するとトランジションの直前でメッセージレイヤが自動的に消去されます。　@ タ trans=crossfade time=1000 msgoff は、@msgoff と @ タ trans=crossfade time=1000 @ 2 行にわけて書いても同じ動作になります。 @ しおり前中制服ポーズ b 無表情 trans=crossafade time=500 msgoff も同じく 2 行にわけて書くことができます。

実際にスクリプト中で msgoff 属性を使うというよりは、自動トランジションを定義する際に msgoff 属性を使用して、背景を変更する際はメッセージレイヤを常に消去する、というような演出をする場合に使うと便利です。

■選択肢

3 章ではかなり簡単にしか説明していなかったので、ここで選択肢について詳しく解説します。

```
*start| スタート
選択して下さい。
@seladd text= 選択肢１ target=* 選択肢１
@seladd text= 選択肢２ target=* 選択肢２
@select

* 選択肢１
選択肢１が選択されました。
@s
* 選択肢２
選択肢２が選択されました。
```

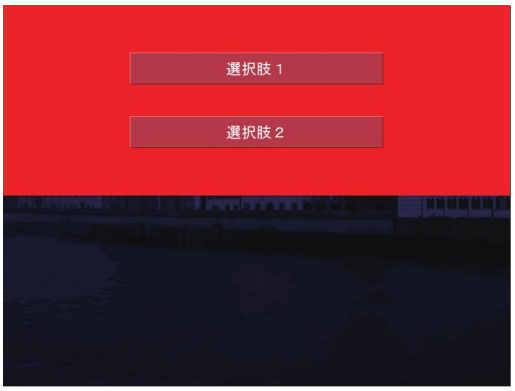
３章の復習になりますが、 seladd タグで選択肢を登録し、 select タグで選択肢を表示することができます。

選択肢には直接関係ないのですが、「 s 」タグでスクリプトの実行を停止できます。選択肢１が選択された際に停止しなければ 選択肢２が選択されました。の部分まで実行されてしまうので @s で停止させています。

```
*start| スタート
@selopt left=0 top=0 width=800 height=300
@seladd text= 選択肢１ target=* 選択肢１
@seladd text= 選択肢２ target=* 選択肢２
@select
```

行数の節約のために * 選択肢１ と * 選択肢２ のジャンプ先のスクリプトは省略しています。動作を確認する場合は先程の * 選択肢１ から 選択肢２が選択されました。までを付け足し

てください。ここから選択肢の説明をしている間は共通です。



選択肢の動作は「 selopt 」タグを使うことで設定できます。 left 、 top 、 width 、 h 属性で「選択肢領域」を指定できます。 left 属性と top 属性でそれぞれ横、縦方向の位置、 top 属性と width 属性でそれぞれ幅と高さを指定します。

選択肢領域とは選択肢を表示する範囲のことです。その範囲に選択肢が等間隔で並べられます。右の画像ではわかりやすいように領域に赤色を塗っていますが、実際には透明で見えません。デフォルトの選択肢領域は画面全体となっています。

選択肢に画像を使うことができます。サンプルの uiimage フォルダの「選択肢 _ normal.png 」「選択肢 _over.png 」「選択肢 _on.png 」をプロジェクトフォルダの「 uiimage 」フォルダにコピーしてください。

```
*start| スタート
@selopt normal= 選択肢 _normal over= 選択肢 _over on= 選択肢 _on
@seladd text= 選択肢１ target=* 選択肢１
@seladd text= 選択肢２ target=* 選択肢２
@select
```

selopt タグの「 normal 」「 over 」「 on 」属性に選択肢に使用する画像のファイル
を指定します。それぞれ通常時、マウスがボタンに乗った時、ボタンがクリックされた時、の画像を指定します。ここで指定した画像の上に seladd タグの text 属性に指定したテキストが描画されて表示されます。

```
*start| スタート
@selopt fadetime=1000
@seladd text= 選択肢１ target=* 選択肢１
@seladd text= 選択肢２ target=* 選択肢２
@select
```

sekiot タグの「 fadetime 」属性で、選択肢を表示する際のフェードの時間を指定します。デフォルトでは 0 でフェードなしですぐ表示されます。選択肢を表示する際のフェードイン、選択後に消去する際のフェードアウトの時間を共通で指定します。

```
*start|  スタート
@selopt normal=  選択肢  _normal over=  選択肢  _over on=  選択肢  _on
選択肢の設定をしました。

@seladd text=  選択肢１  target=*  選択肢１
@select

*  選択肢１
選択肢１が選択されました。

@seladd text=  選択肢２  target=*  選択肢２
@select

*  選択肢２
選択肢２が選択されました。
```

selopt タグで指定した設定は、それが変更されるまで変わりません。最初に１度だけ指定しておけばそれ以降の選択肢で使用されます。 @selectごとに @selopt タグを使う必要はありません。上のスクリプトでは、最初の @selopt による画像が両方の選択肢で使用されます。

■後書き

凄い中途半端なところでぶった切りますが、ここでタイムアウトです。現在、頒布前日の 12 月 30 日 22 時を回ったところです。

最初は初心者向けに書いていたのですが途中から何が何やら、どんどん残念な方向に。そこらへんの言い訳はブログや Twitter を見ると残ってるかもしれません。ここでも申し開きしたいところですが我慢します。次の夏コミにはちゃんとしたものを出したいものです。

まだ準備できていないのですが、冬コミ後にはこの本で使う SDK を公開します。遅くとも 1 週間以内になんとかします。そちらの方に説明書と言う名の補足が付いている可能性があるのでチェックするようにお願いします。本文のどこかにも書いてありますが、本書のサポートウェブページは、 http://biscrat.com/book/kagex/support/ となります。

次の章は大幅に方針転換しまして、 KAG を使ってる方向けに、 KAGEX 使う上で参考になりそうな話を詰め込んでおきます。少しでも役立つように思いついたところを片っ端から書いていきます。完全に今から考えるのでひどい事になるかもしれませんがご了承ください。

7. おまけ

■オブジェクトについて

本編の最初の方で stage 、 char 、 layer 、 event というタグを直接使ってましたが、実際には使わないでください。ほぼ間違いなく省略記法で使います。初心者向けにタグの説明したいということで丁寧にやってみましたが今振り返ると失敗でした。

それらは簡単にするために「画像を表示するためのタグ」として紹介しました。実際は stage は舞台オブジェクト、 char はキャラクタオブジェクト、 layer は環境レイヤオブジェクト、 event はイベントオブジェクトを操作するためのタグです。それに加えて env が環境オブジェクトを操作するタグ、 bgm が BGM オブジェクトを操作するタグ、 se が効果音オブジェクトを操作するタグです。

KAGEX ではデジタルノベルで使用される概念を「オブジェクト」としてまとめています。 KAG だとシステム的にもっと低レベルで、 image タグでレイヤを直接扱ったりしました。 KAGEX ではそんなことはしません。

特にキャラクタが例として分かりやすいですが、 char タグで操作できるキャラクタオブジェクトには、キャラクタとして必要な操作がまとまっています。立ち絵表示、ボイス再生、表情画像表示などがまとめて char タグ１つでつかえます。内側では何枚ものレイヤと再生バッファを扱っていて、 KAG のプリミティブな操作と比べるとかなり高レベルな実装になっています。

ところで低レベル、高レベルというのはプログラム的な意味での言葉なのであしからず。低レベルな処理というのは、色々できますが色々やらないと駄目という意味で、高レベルな処理というのは用途は特定されてるけども便利な機能が簡単に使えるという感じの意味です。

■補足

立ち絵については結局表情結合済みの立ち絵しか扱えていません。少し規模が大きくなると、表情部分は分離することが多いと思います。それについての説明は開発ツールに付属するはずなのでそちらを参照してください。

本編でアクション定義について説明できなかったのですが、今回しっかり調べてみたところ思った以上に複雑で説明端折ってしまいました。こちらも開発ツールの方に説明つくと思います。

■シナリオスクリプト / システムスクリプト

本編の説明は、ほぼシナリオスクリプト部分の説明です。 KAGEX 使う上ではゲーム本編中の「シナリオスクリプト」と、タイトル画面やセーブ / ロード画面、オプション画面などの「システムスクリプト」はわけて考えたほうがわかりやすいです。システムスクリプトについては、この後の章で扱う予定でしたがカットです。

シナリオスクリプトは、実は本編で説明されている知識だけで十分だったりします。シナリオ中ではテキスト表示と立ち絵の服装や表情変更、加えてたまに効果音やBGM を鳴らすくらいしかしません。それだけ知ってれば作れます。

■マクロ

本編でマクロの説明をしていないんですが、 KAGEX の素の部分だけで既に必要なだけまとまっています。 KAGEX で何本かゲーム作ってますがマクロなんてほとんど使いません。

マクロ使うとしたらフィニッシュ時のフラッシュのような定型のエフェクトつけるくらいですね。

@endtrans についてはマクロにしてしまうと便利です。立ち絵切り替えや背景切り替えでは自動トランジションが登録できるのでいいのですが、 @endtrans では自動トランジションが定義できません。

```
@macro name=et
@endtrans 背景トランジション *
@endmacro
```

これだけでもいいです。 @endtrans のかわりに @et を使えば「背景トランジション」を自動トランジションのように使えます。

マクロの定義は「 sysinit 」フォルダの「 macro.ks 」でするようにしてください。少し「 first.ks 」を覗いてもらえれば分かりますが @call storage=macro.ks となっています。KAG でもマクロ定義スクリプトを複数回読み込んでいる初心者の方がいるんですが、macro.ks は起動直後に１回しか読み込まれない専用スクリプトファイルとして最初から準備してしまっています。

■envinit.tjs

シナリオスクリプトは非常に簡単なのですが、その前段階としてのシステムスクリプトが必要になります。 KAG から移行する上でこれが壁になります。本来は「 envinit.tjs 」というファイルに舞台オブジェクトやキャラクタオブジェクトの内容について、TJS の辞書配列で定義していきます。これが難解なため KAGEX 難しいというイメー

ジがついてしまうのではないでしょうか。実際はそこまででもないんですが、リファレンスが本当に皆無なために最初は引っ掛かります。

本書では envinit.tjs の代わりに envinit.otjs を書いています。 envinit.tjs を直接書くよりは、 config.tjs を書くのに似ていて簡単だとおもいます。それでも面倒なのでツールでできるようにする予定です。 SDK がリリースされたら説明書を参照してください。

本書ではもう１つの簡略化として、背景や立ち絵などの画像の名前形式を固定化してしまっています。本来はこの名前のルールからして定義しないと駄目なのが初心者お断わりに拍車をかけています。こんなものは固定でいいはずです。 SDK を使う場合は本編に書いてあるルールに従って頂ければ OK です。 autosetting.bat というのを何度か本編で使いましたが、それでファイルを検索して envinit.tjs に変換しています。

■選択肢

選択肢で seladd 、 select 、 selopt を紹介しました。似たような選択肢しか使わないゲームならこれで十分です。気に入らなければ KAG と同じく button タグや link タグが KAG と同じように使えるので活用してください。 button タグについては graphic 属性で通常時、マウスオーバー時、マウスクリック時が並んだお馴染みの画像も指定できますが、選択肢と同じく normal 、 over 、 on 属性でバラバラにした画像も読み込めるので便利です。

■TJS 的な話

KAG では image とか trans とか playbgm とか、とにかく色々なタグを使っているとおもいます。 KAGEX では９割方 char 、 stage 、 event 、 layer 、 env 、 bgm 、 つのタグで何とかなります。タグが少ない代わりに属性の種類が多いです。このオブジェクトを操作する属性のことをコマンドなんて言ったりします。名前はどうでもいいですが、 system フォルダを commands で grep するとコマンドの動作が追えます。

KAG だと tagHandlers を見ておけばそれぞれのタグがどんな動きをしているかが分かったのですが、 KAGEX ではオブジェクトごとに属性がまとまっています。例えばキャラクタオブジェクトについては「 KAGEnvCharacter.tjs 」の%[var charCommands = %[以下に char タグで使える属性の動作が書いてあります。これだけでも知っていると KAGEX の TJS を読みたい場合は楽になるかもしません。

ついでに「 KAGEnvImage.tjs 」の%[var commands = %[以下は画像を扱えるオブジェクト (stage 、 char 、 event 、 layer) で共通に使える属性の動作となっています。

■変数

本編で一切触れていないのですがゲーム変数 f 、システム変数 sf 、一時変数 tf は KAG と同じく好きに使ってもらって大丈夫です。 if タグでの条件分岐なども必要であればご自由にどうぞ。最近そういったフラグ管理が必要なゲームはあまり作ったことがないんですが。ほぼ一本道か選択肢何個か選んで〜程度のゲームばっかり作ってます。

■システムスクリプト

SDK ではシステム部分も簡単に使えるように定型化しています。ゲーム起動後は「 first.ks 」から始まりますが、少し追ってみると @eval exp="biscrat_handler.callExtraConductor('title.ks')" というのがあります。 title.ks の最初に call タグで飛んでいる感じだと思ってください。 SDK のシステムスクリプトでは exp=biscrat_handler. というのを多用します。 KAG で少し tjs の機能を使っていた方は exp=kag. をよく使ったと思いますが、それと同じようなものです。

title.ks の方を見てみると「状態の初期化」→「タイトル画面の表示」という流れになっています。初期化の方はそのままにしておいてください。ゲームエンディング後にタイトル画面に戻ってきても困らないように色々と初期化しています。

思い出しましたが、エンディング後にタイトルに戻ってくる際には @eval exp="biscrat_handler.funcObject.goToTiltle()" で戻ってきてください。 next タグなどで直接 title.ks に飛んでくると困ります。

first.ks の初期化で、 @position については好きなように直してください。本編では position タグ自体の説明のために 01.ks の冒頭でやっていますが、 title.ks で初期化と一緒にメッセージレイヤの設定もしてしまったほうが便利だと思います。

KAGEX ではメッセージレイヤにシステムボタンを設置できます。メッセージレイヤでスキップやクイックセーブができると便利ですよね。 sysbutton タグを使ってそれができます。

```
@sysbutton normal= スキップ _normal over= スキップ _over on= スキップ _on x=100 y=40 exp=biscrat_handler.funcObject.switchSkip() nostable
```

button タグによるボタンと同じく、システムボタンはメッセージレイヤ上に配置されます。 normal 、 over 、 on 属性にボタン画像を指定します。 x,y 属性でメッセージレイヤ上の位置を指定します。 locate タグは使わずに属性として使えます。 nostable 属性は、 true を指定すると文字を表示している最中などにもボタンがクリックできるようになります。 storage 属性と target 属性でジャンプ先を指定することはできないので、 exp 属性にクリック時に実行する TJS 式を指定します。ここに書く TJS スク

リプトは準備してあります。

biscrat_handler.funcObject.switchSkip()でスキップの開始、biscrat_handler.funcObject.switchAutoMode()でオートモード開始、biscrat_handler.funcObject.switchMessage()でメッセージレイヤの非表示、biscrat_handler.funcObject.switchHistory()で履歴の表示、biscrat_handler.funcObject.goBackHistory()で通過記録をたどる、biscrat_handler.funcObject.exit()で終了、biscrat_handler.funcObject.quickSave()でクイックセーブ、biscrat_handler.funcObject.quickLoad()でクイックロード、biscrat_handler.funcObject.playCurrentVoice()で現在のボイスの再生、biscrat_handler.funcObject.goToSave()でセーブ画面へ、biscrat_handler.funcObject.goToLoad()でロード画面へ、biscrat_handler.funcObject.goToOption()でオプション画面へ、biscrat_handler.funcObject.goToTitle()でタイトル画面へ、の機能が使えます。

これだけでも一般的なシステムボタンは作れると思います。ここらへんのサンプルは、 SDK 公開には間に合わないと思いますが、後ほど公開するので参考にしてください。 title.ks の @position と @syncmsg の間に sysbutton タグで追加してしまうのがお勧めです。

syncmsg タグはメッセージレイヤのみ表ページから裏ページへのコピーを行います。 position などでメッセージレイヤの設定をした場合は、裏ページも更新しないとたまにバグるので @syncmsg はメッセージレイヤの設定をした後にセットとして書いておくのを推奨します。

実際のタイトル画面は * タイトル画面表示のところから書いてください。メッセージレイヤの 1 番あたりに、 position 、 button 、 link タグを使ってタイトル画面を表示されるのを想定しています。ここらへんは KAG と変わらないです。初期状態では画面を表示 せずにすぐに *start ラベルに飛んで、そこからゲーム本編の 01.ks に飛んでいます。

ゲーム本編にジャンプするときは、 next タグではなく return タグを使ってください。タイトル画面は call タグのようなもので飛んできています。これはセーブ / ロード画面やオプション画面を作る際にも同じです。 call から戻るために return を使います。

そのセーブ、ロード、オプション画面ですが、「 sysscn 」フォルダの「 system.ks 」に作ることになっています。それぞれ *save 、 *load 、 *option ラベルで表示させます。タイトル画面と同じように、 button タグや slider タグなどを使って頑張って表示させることになります。 KAGEX だとシステム画面は TJS を使って書くのがお勧めです。 slider などが最初から使えるので KAG よりはマシですが、タグを使ってシステム画

面を構築するのは同じくらいしんどいです。ここらへんのサンプルも後で公開できればするかもしれません。

一番のお勧めは自分に発注しちゃうことですねー（笑）システム画面の組み込みや KAGEX 導入のサポートくらいなら同人で 1 万円程度でも引き受けてます。シナリオスクリプト部分は誰でもできるレベルで簡単なので、システム部分のみ人に任せるというのはアリだと思います。

■リリース

リリース時は KAG と特に変わりません。リリーサ使ってまとめてしまえば OK です。

■メニューとか

デフォルトの KAGEX と比べてもゲーム内メニューは拡張してあります。表示 / 非表示あたりもツールで切り替えできるようになる予定なのでもう少し待ってください。

■ボイス

ちゃんとシナリオ順に番号振って、セリフごとに自動的に再生させるのがお勧めです。台本出力の時にどうせ番号ふるし……ということなら毎回 voice 属性でちゃんと番号振ってもいいいかもしれません。今まで KAGEX 使っていて 1 度だけボイス番号がずれるバグに遭遇しました。何が起こったのかわからないですが多分 2 度と再現できないので気にしてません。選択肢などで分岐する場合はボイス番号がずれると思うので指定忘れないようにしてください。ボイスのデバッグ時はメニューにある倍速再生がかなり便利です。しっかり聞き取れなくても何かおかしいってのは早く再生してもわかるものです。 1 つ 1 つちゃんと聞く時間があればそれでいいんですが。

■動画再生

KAG だと何かタグがたくさんあってめんどくせえのですが、 KAGEX では環境レイヤの movie 属性使ってしまうと簡単です。 @layer name= 動画 movie=movie.mpg のようになります。再生待ちには @wv を使います。

```
@layer name= 動画 movie=movie.mpg
@wv
@ 動画 delete
```

これだけです。

■ウェイト

作者がゲーム中に待たされるのが大嫌いなので本文中で使ってませ

んが KAG の@waitも普通につかえます。それに加えて、大抵の同期動作のタグでは wait 属性が使えます。属性値に時間を指定すると、同期動作終了後にその時間だけウェイトが入るようになります。 @街昼 fade =1000 sync wait=1000では、フェード終了後にさらに 1 秒ウェイトされることになります。wait 属性をつけると sync 属性も自動的に入るのでそちらは省略してしまっても大丈夫です。@ 街昼 fade=1000と @wait time=1000の 2 つに分けると飛ばすのに 2 クリック必要ですが、 wait 属性で 1 つにまとめると 1 クリックで飛ばせます。まとめてあげてください。

■[タグ] について

本書では [] で囲んだタグを一切使っていません。 KAGEX では改行や空行での自動改行や自動ページ送りがあるので [] を使うとちょっとめんどくさいことになる場合があります。 @ で始まるコマンド行ならその心配が無いので自分の場合はこちらに統一しています。

■ラインモードについて

本書では、テキスト中の改行はそのまま改行、空行はページ送りとなっています。ここらへんの動作は「ラインモード」として設定できるのですが、細かいことはツールに付属する説明書をみてください。作品によっては変える必要があるかもしれません。改行ごとにクリック待ちをさせるラインモードなどもあるのですが、全く推奨しません。できるだけクリックさせて欲しくないです。

■テキスト全画面表示について

KAGEX はテキストを画面全体に表示するゲームにはあまり向いていません。まあできないことはないですが、名前欄の機能は使えなくなってしまうのではないでしょうか。

■トランジション

KAG の trans タグは使わないでください。管理が面倒になるだけです。

■時間切れ

作者のブログ (http://kasekey.blog101.fc2.com/) でもいくらか KAGEX 関連の記事があります。そちらもウォッチしてみると参考になるかもしれません。メール (sakano@biscrat.com) で質問も一応は受け付けてます。 Twitter(https://twitter.com/kasekey) でもどうぞ。

吉里吉里　/KAGEX　講座
～デジタルノベル製作～

発行日　2011　年 12　月　31　日　初版発行
著者　sakano

Email　sakano@biscrat.com
Tel　080-5095-02037
Web　http://biscrat.com/
Twitter　http://twitter.com/kasekey/

ご意見ご感想お待ちしております。