卒 業 研 究 報 告

題　　目

Web ベースの言語トレーニングゲームの開発

指 導 教 員

綿森道夫 助教授

報 告 者

学籍番号: 1030231
氏名:　　吉村 和倫

平成 15 年 1 月 27 日

高知工科大学 電子･光システム工学科

目次

第 1 章　はじめに

アニメーション作成ソフトである Flash の学習の過程で、アクションスクリプトという言語を用いてアニメーションだけではなくゲームも作ることができるということを知り、卒業研究にこのテーマを選びました。

本研究では、まずゲームのシナリオ考案から始まり、キャラクター、建物などのグラフィック作り、プログラム作成などのゲームを作り始めてから完成するまでの開発手順について学習していきたいと考えています。

第 2 章　目的

Web 用のアニメーション作成ソフトである Flash の使用法や、Flash の言語であるアクションスクリプトについて学習する。それらを学んだ上で、ゲーム感覚でアクションスクリプトについて学習できるゲームを作成する。

そして、出来上がった作品を実際に Web 上にアップしておき、いつでも見ることができるような形にすることを目的とする。

第 3 章　使用ソフトの解説と特徴

3.1　macromedia Dreamweaver

Dreamweaver はホームページを作成するソフトです。卒業研究でホームページを作成し、Web 上にアップする時にこのソフトを使用しました。

Dreamweaver を使う利点としては、ホームページのイメージを修正する時に、メニュー操作だけで Fireworks を呼び出すことができ、Fireworks で編集した結果がそのまま Dreamweaver の編集画面にも自動的に反映されるということ、またホームページに Flash の作品を載せたい場合は、ボタン操作で簡単に貼り付けることができるという点だと考えます。

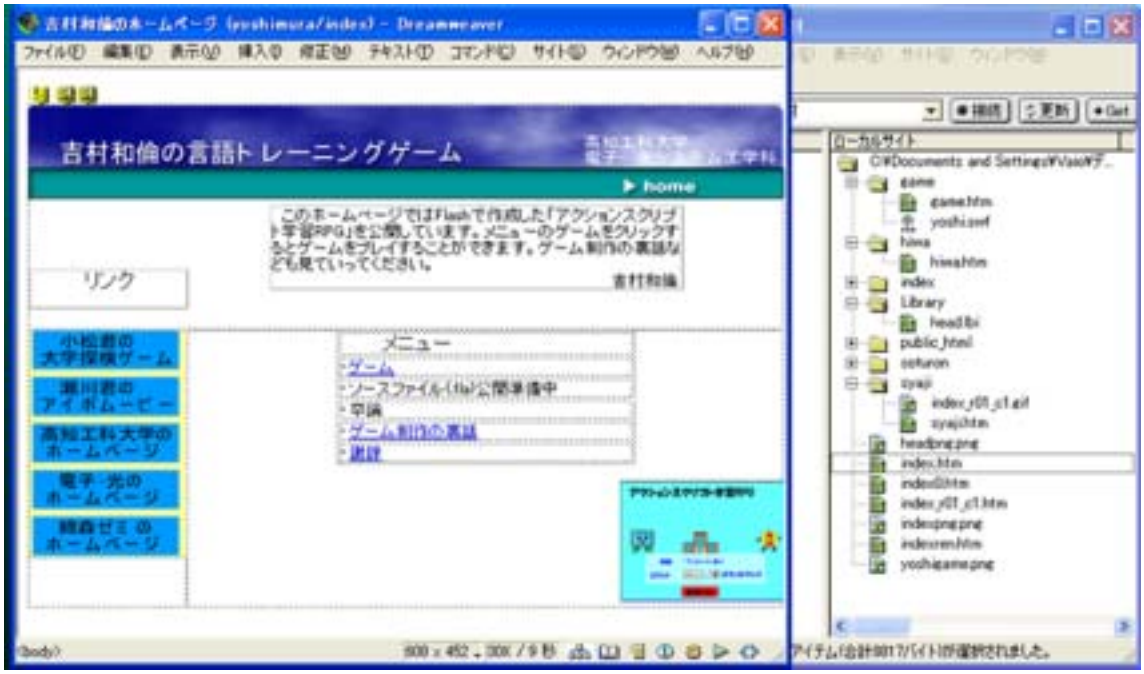


図 3－1　Dreamweaver で作成したホームページ

3-2　macromedia Fireworks

Fireworks は Web 上に配置する JPEG、GIF、PNG ファイルなどを作成するためのソフトです。卒業研究ではホームページに貼り付ける画像を作成するのにこのソフトを用いました。

このソフトの利点は作成した画像のサイズや形式の変換が容易であるということ、同時に Fireworks では Dreamweaver 形式のファイルを直接書き出すことができるので、画像のみならず HTML ソースも書き出すことができるということだと考えます。

またホームページのみならず、この論文で使用されている画像は Fireworks で png ファイルに変換して貼り付けています。

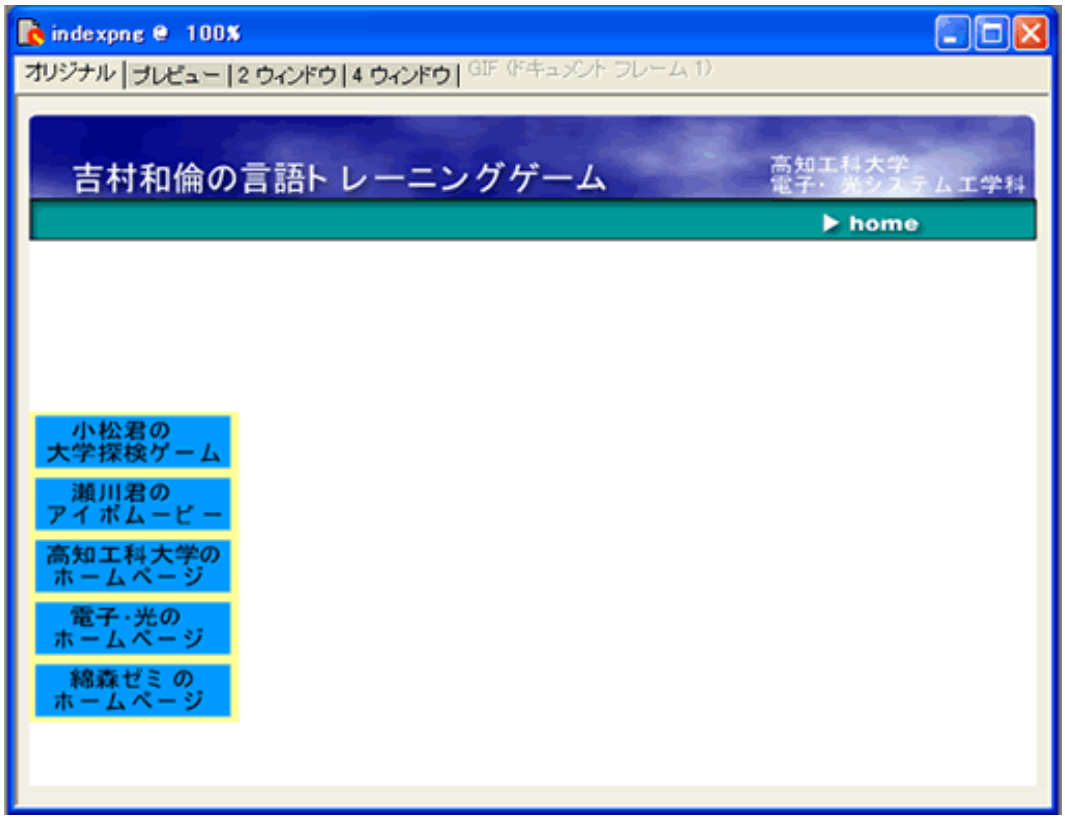


図 3－2　Fireworks で作成したトップページの画像

3-3　macromedia Flash

Flash は Web 用のアニメーション作成ソフトとして使用されます。
またアニメーション作成だけではなく、ゲーム作成や画像編集にも使用されます。Flash で作った作品はパブリッシュ(Web 上で公開できるように書き出す作業)を行うと、Web 公開用のファイル(Flash Player ファイル)を生成し、Web 上で見ることができるようになります。

Flash で描いた絵はベクター画像といい、形や塗りの情報をもっており、どんなディスプレイでもリアルタイムに再計算して適した画像を生成するので、ペイントソフトに比べてデータサイズが小さくなるという利点を持っています。

また、パブリッシュを行って exe ファイル(Windows プロジェクター)形式で書き出すと、パソコンに Flash がインストールされていない環境でも exe ファイルで作品を見ることができるという利点があります。

ゲームの制作ではすべて、Flash を使って行いました。

第 4 章　作品の概要

4.1　作品のイメージ

- ロールプレイングゲーム
- ゲームをしながらアクションスクリプトの学習ができる
- アクションスクリプトに関するクイズイベントに正解しないと先に進めない
- 1 時間ぐらいで終わるゲームにしたい
- スクリプトに関する知識を店で買えるようにしたい
- 敵キャラクターに動きをつける

4.2　作成手順

下に示すのは作成手順のフローチャートです。

1　ゲームのプレイ時間をどのくらいにするのか考える

2　アクションスクリプトの種類と意味を調べる

3　キャラクター考案

4　ゲームの主人公や敵キャラクターの能力値の設定

5　アイテムをどうするのかを考える

6　キャラクターに動きをつける

7　イベントで出題するクイズを考える

8　マップや建物などのシンボル作成

9 プログラムの作成

10 テストプレイ

では、順番に説明していきます。

1　ゲームのプレイ時間をどのくらいにするのか考える

まずゲームを作るにあったて最初に考えたことはどれくらいの時間で終わるゲームにしたいのかということです。Web でできたゲームを公開するということもあったので、1 時間ぐらいで終わるロールプレイングゲームにしようと考えました。

2　アクションスクリプトの種類と意味を調べる

このゲームの特徴はゲームをしながらアクションスクリプトの言語について学ぶことができるようにしたかったので、次にアクションスクリプトの種類や意味を文献などで調べました。スクリプトの意味や種類を調べ、どういったものが始めのほうで説明されているのか、Flash でよく使われるスクリプトは何かを調べました。よく使用されるスクリプトとしてどんなものがあるかの一例を次に示します。

- on() --- ボタンアクションを定義するもの
- onClipEvent() --- クリップアクションを定義するもの
- if --- 条件が満たされた時、記述されたスクリプトを実行
- for --- 決まった回数繰り返し処理を行う
- gotoAndPlay() --- 指定されたフレームに移動しそこから再生
- gotoAndStop() --- 指定されたフレームに移動しそこで停止
- _root --- ムービークリップが属する大元のタイムラインを参照する
- _parent --- 1 つ上の階層、すなわちそのムービークリップが置かれたタイムラインを参照する
- function --- 複数のスクリプトを組み合わせ、ユーザー独自のメソッドを定義する
- Math オブジェクト --- 数値演算を行う
  Math.random() --- 0 以上 1 未満の乱数を発生
  Math.floor() --- 小数点以下を切り捨てて整数化する

3　キャラクター考案

　次に、戦闘シーンで登場する敵キャラクターを考えました。
敵キャラクターはタブレットを使って作成しました。
図 4-2-1 にタブレットを使って描いたキャラクターを示します。

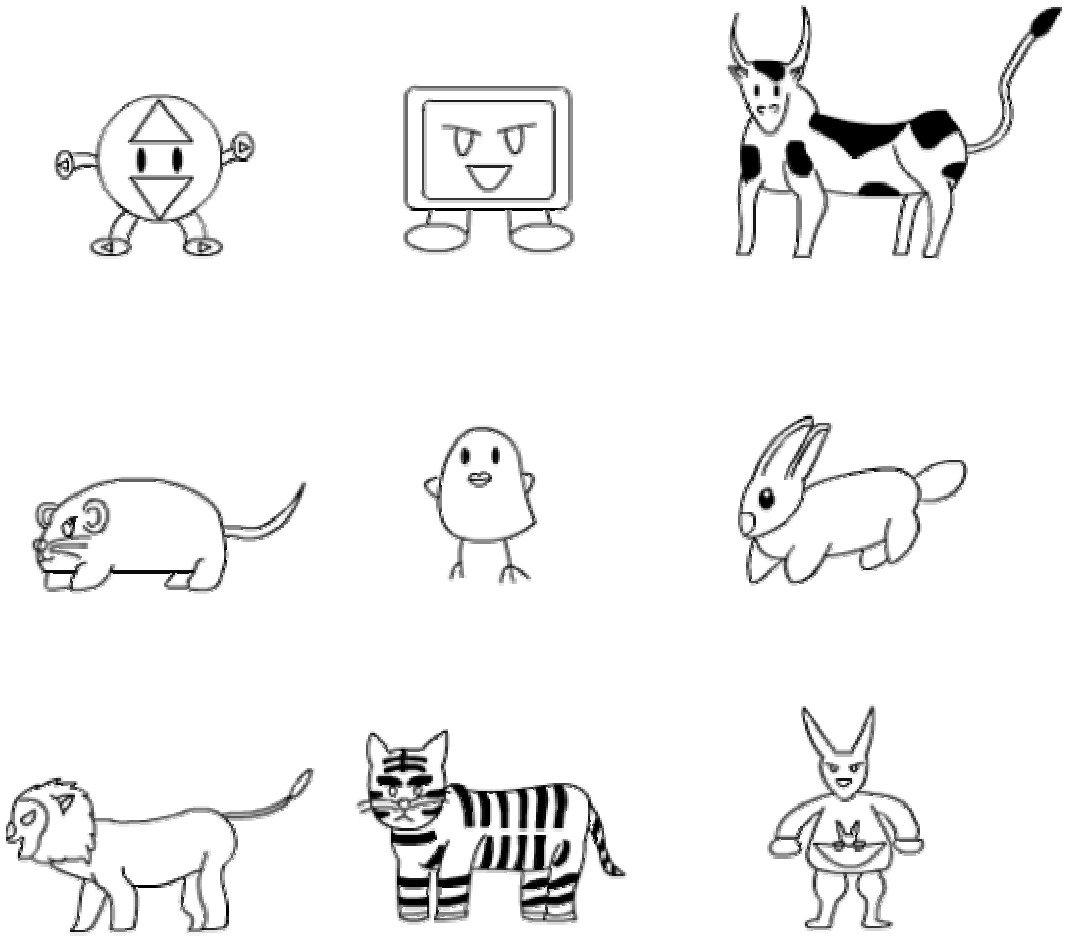


図 4－2－1　　敵キャラクターの下絵

4　ゲームの主人公や敵キャラクターの能力値の決定

　はじめに 1 時間ほどで終わるゲームにしたかったので主人公のレベルの最大を 13 まで上がるようにしました。図 4-2-2 に主人公、図 4-2-3 に敵キャラクターの能力値の一部を示します。

- Lv　　--- 主人公の現在のレベル。最大で 13 まで。
- HP　　--- 体力を意味する。0 になると気絶する。
- 攻撃力　--- 攻撃力が高いと敵に大きなダメージを与える。
  ただし、敵の守備力のほうが自分の攻撃力より大きいと敵にダメージを与えることができない。
- 守備力　--- 敵の攻撃から身を守る力。
  守備力が高いと敵からダメージを受けにくい。
- 経験値　--- 敵を倒すと手に入る。
- 次のレベルまで ---次のレベルまで必要な経験値。
- お金　　--- 現在持っているお金。お金がないとアイテムなどを買うことができない。

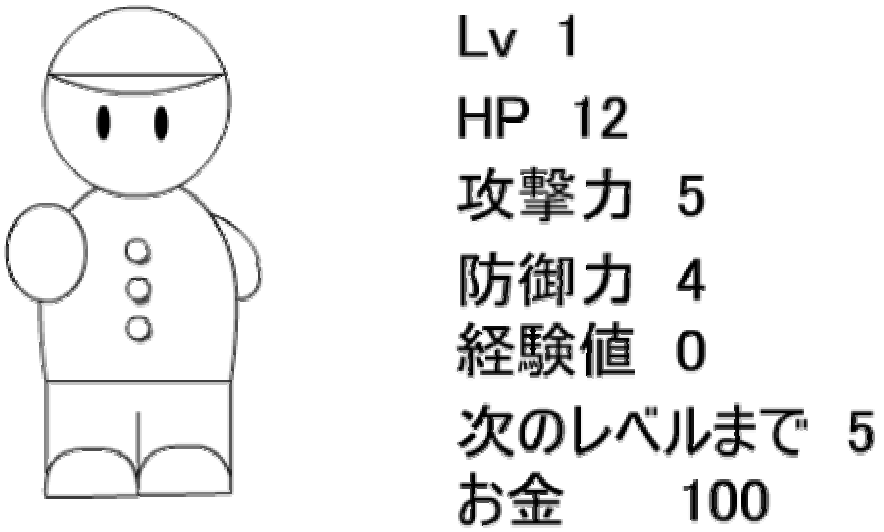


図 4－2－2　主人公の能力値(ゲームスタート時)

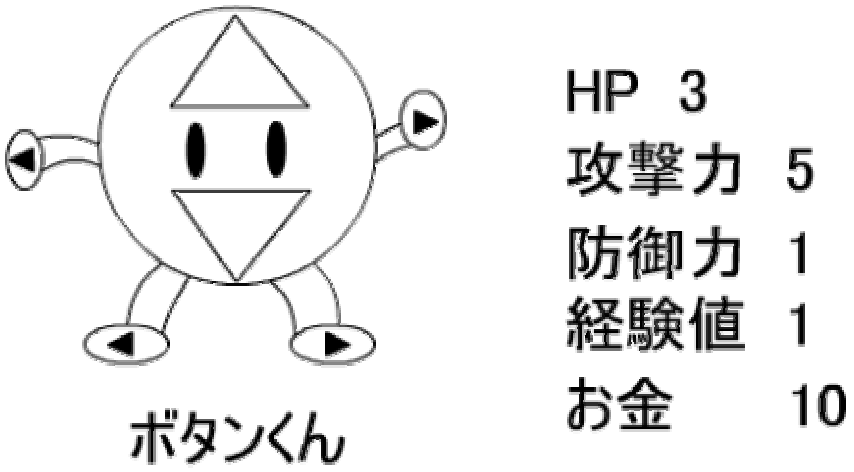


図 4－2－3　敵キャラクターの能力値(ボタンくんの場合)

5　アイテムをどうするのかを考える

　次にアイテムをどういったものにするのか考えました。ゲーム中に使用できるアイテムを、「回復」、「武器」、「防具」の 3 種類に決めました。アイテムを大きく 3 種類に分けた後、その中に属するアイテム名や使用時の効果を考えました。

アイテムの種類

「回復」--- 自分の HP を回復させる。一度使うとなくなる。
(例)　コーヒー、おにぎり、健康ドリンク　など
「武器」--- 装備すると自分の攻撃力が上がる。
(例)　筆記用具、ケイタイ電話　など
「防具」--- 装備すると自分の防御力が上がる。
(例)　メモ帳、参考書　など

アイテムの効果

(例)コーヒー　--- 自分の HP を 10 回復させる。
(例)筆記用具　--- 装備すると攻撃力が 5 上がる。
(例)メモ帳　--- 装備すると守備力が 3 上がる。

6　キャラクターに動きをつける

主人公はマップ移動時に、敵キャラクターは戦闘シーンで動くようにするために Flash で動きのあるムービークリップを作りました。

主人公の動きの作成方法から見ていきます。

まずはじめに、主人公に色を塗ります。

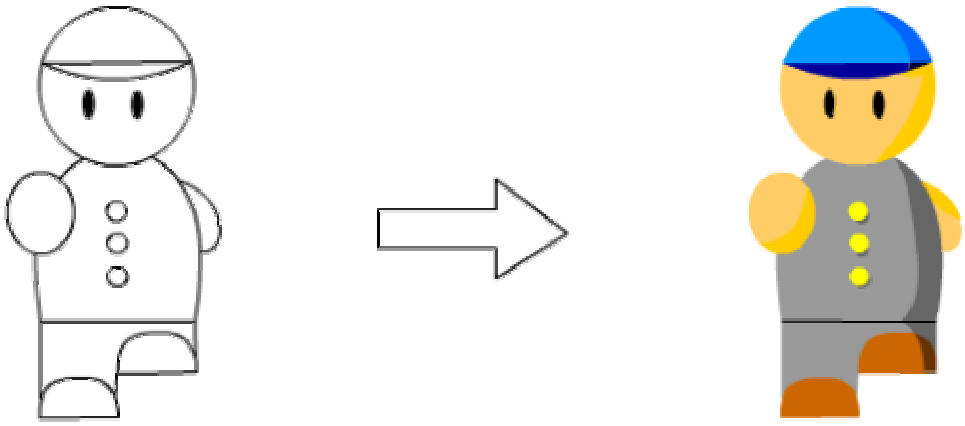


図 4－2－4　キャラクターに色を塗る

次に色を塗ったキャラクターをムービークリップに変換します。

「挿入」メニューの「シンボルに変換」を選びます。

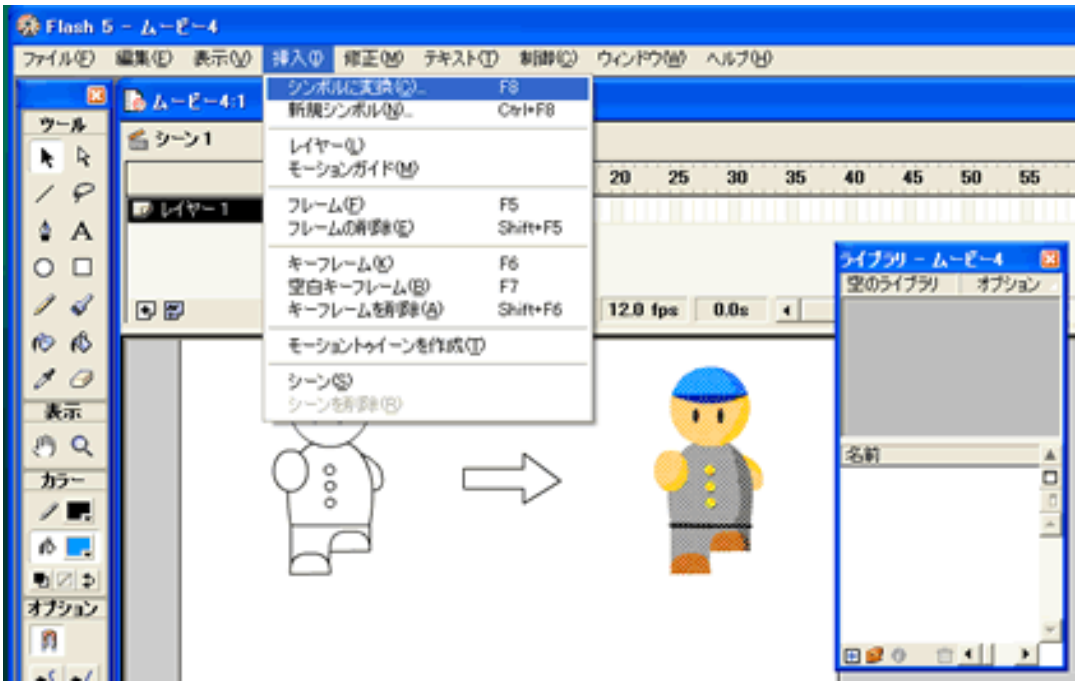


図 4－2－5　シンボルに変換

「シンボルプロパティ」の「タイプ」で「ムービークリップ」を選び、名前を「キャラクター1」とします。

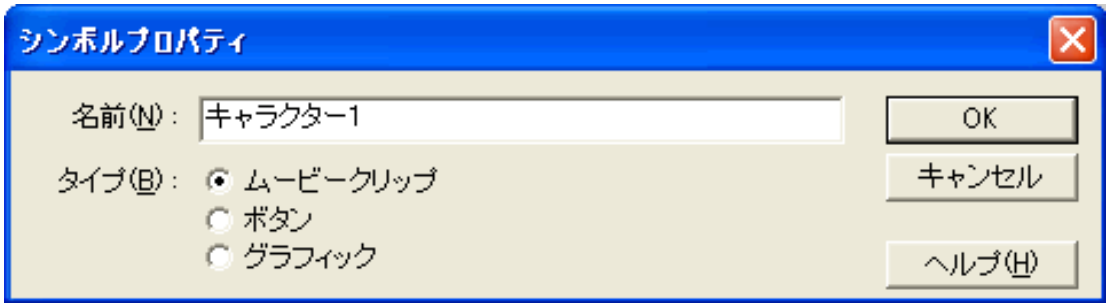


図 4－2－6　シンボルプロパティ

シンボルに変換すると「ライブラリ」ウィンドウの中に「キャラクター1」のシンボルが入ります。「ライブラリ」ウィンドウのプレビュー画面をダブルクリックし、シンボル編集画面に切り替えます。

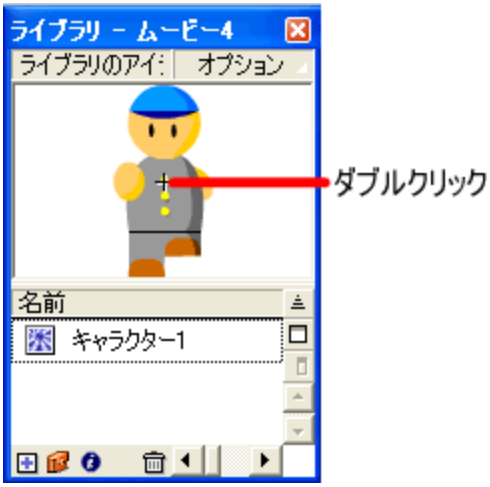


図 4－2－7　ライブラリ

シンボル編集画面で「レイヤー1」の 6 フレーム目で「空白キーフレームを挿入」を選びます。

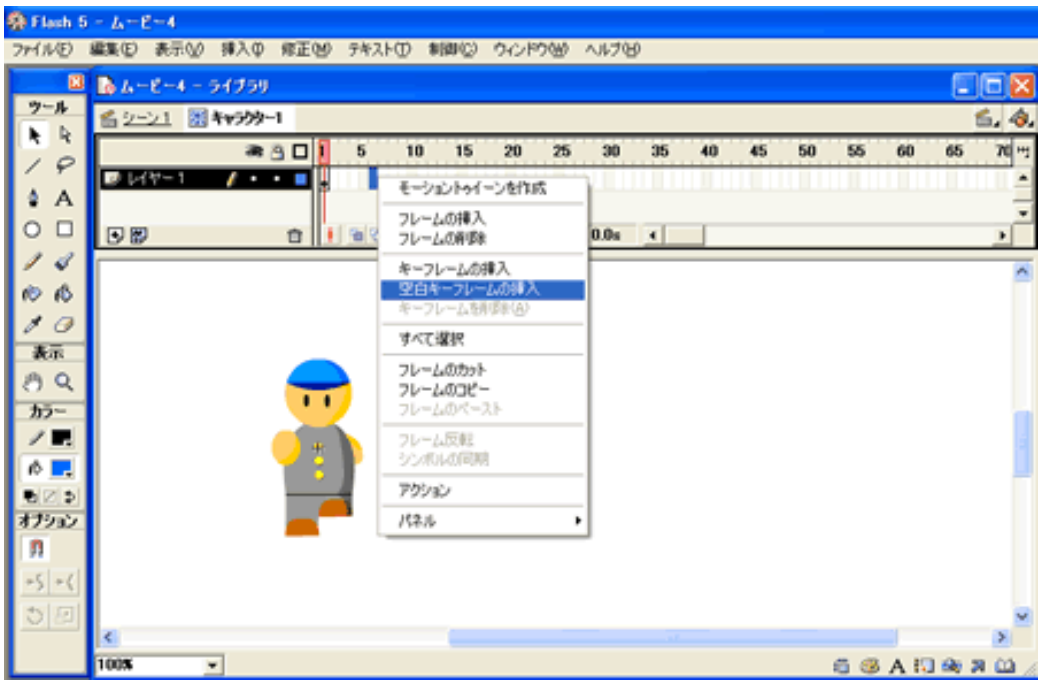


図 4－2－8　「空白キーフレーム」の挿入

6 フレーム目に 1 フレームと手足が逆になった絵を描きます。
絵を描いたら 10 フレーム目で「フレームの挿入」を選びます。

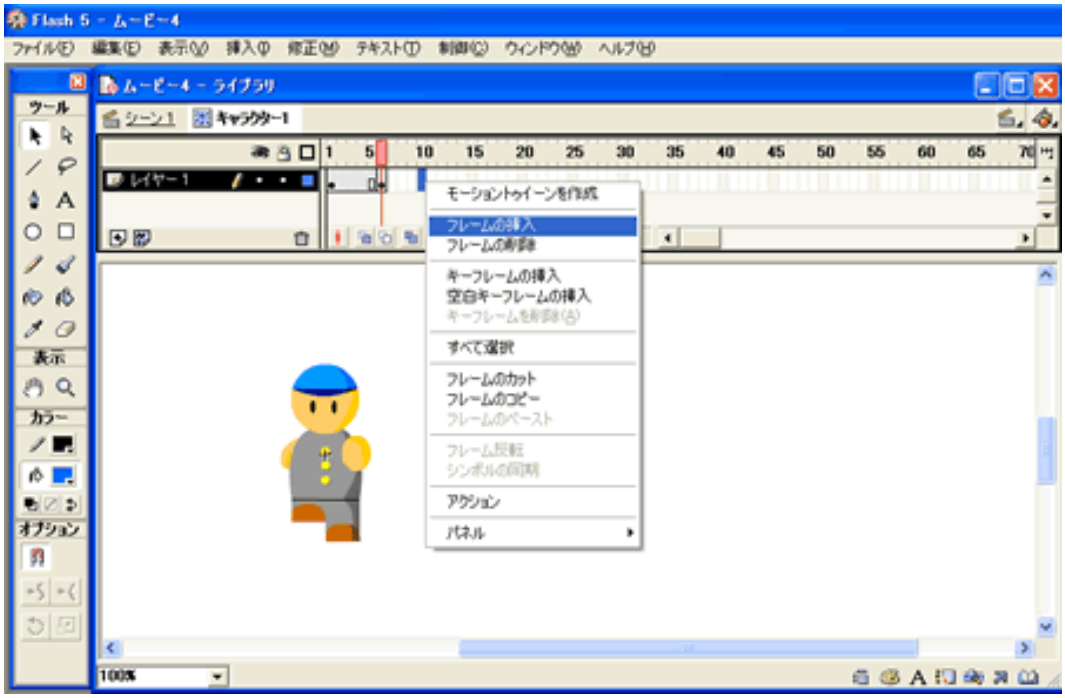


図 4－2－9　「フレームの挿入」

絵が出来上がったら「シーン 1」に戻り、「制御」メニューの「ムービープレビュー」で再生します。ムービークリップはメインのムービーの動きには関係なくアニメーションを開始するので、1 フレームだけの場所に配置して「ムービープレビュー」で再生してやると動いているように見えます。同じようにして、後ろ、右、左向きのシンボルを作っていきます。

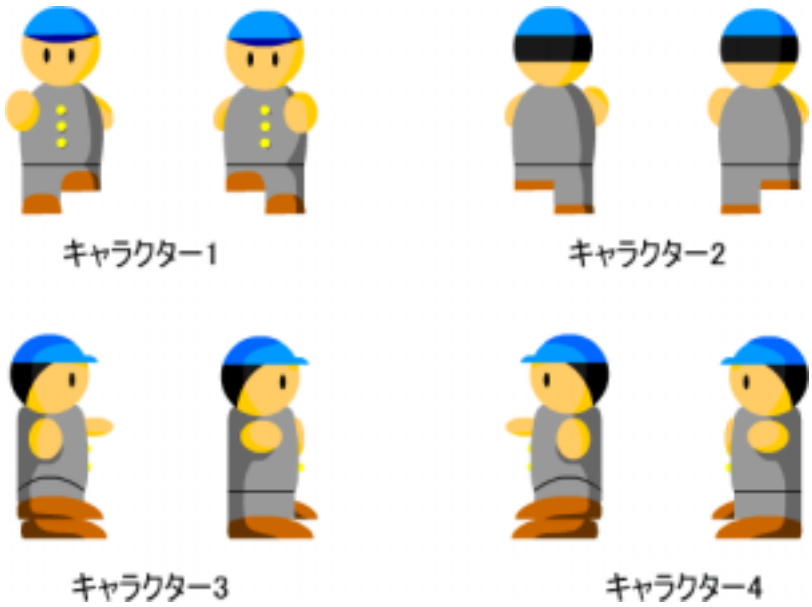


図 4－2－10　キャラクターの動きのシンボル

前後左右それぞれの動きのあるムービークリップを作ったら次はそれらを 1 つのムービークリップにします。「挿入」メニューの「新規シンボル」を選び、「シンボルプロパティ」の「タイプ」で「ムービークリップ」を選び、名前を「キャラクター」とします。

ムービークリップ名、「キャラクター」の中に「キャラクター1(前向き)」、「キャラクター2(後ろ向き)」、「キャラクター3(右向き)」、「キャラクター4(左向き)」のムービークリップを配置します。

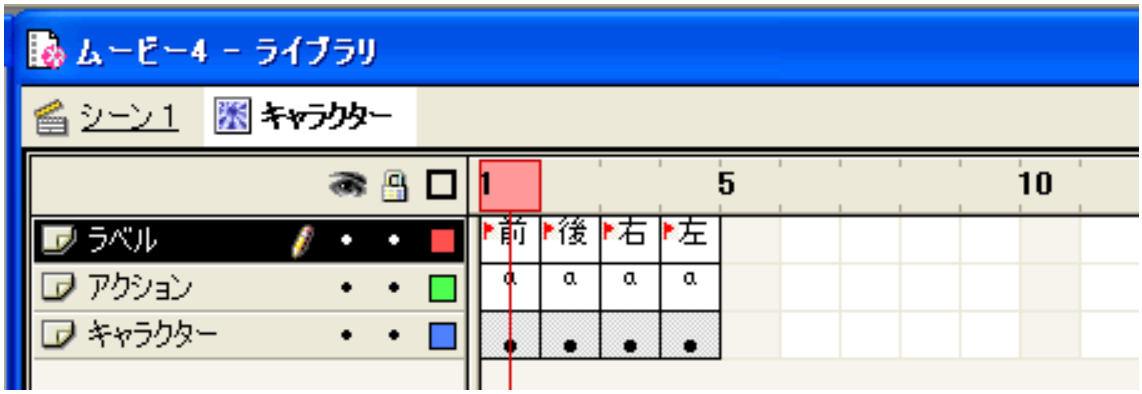


図 4－2－11 ムービークリップ「キャラクター」のタイムライン

図 4-2-11 のレイヤー名「キャラクター」の 1 から 4 フレーム目に「キャラクター1」から「キャラクター4」までを配置します。

レイヤー名「アクション」の各フレームの “ a ” はそのフレームにアクションスクリプトが記述されていることを示します。

図 4－2－12　アクションパネルの表示の仕方

アクションパネルを表示する場合は「ウィンドウ」メニューの「アクション」を選びます。図 4-2-12 の「アクション」のレイヤーには 1 から 4 フレーム目まで、すべて”stop();”が記述されており、ムービープレビューすると 1 フレーム目で止まったままの状態になります。

ここで作った、ムービークリップ「キャラクター」はマップ移動の際で使用します。

敵キャラクターの動きについてもキャラクターの動きと同様に作ります。

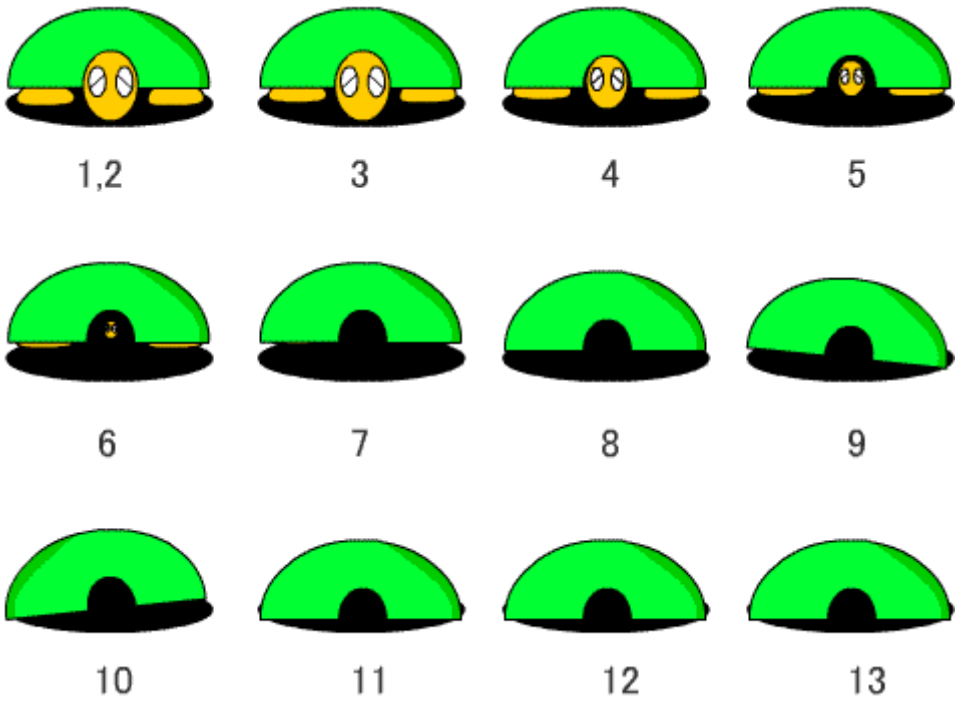


図 4－2－13　モンスター「もしもしカメ」の動き(数字はフレーム番号)

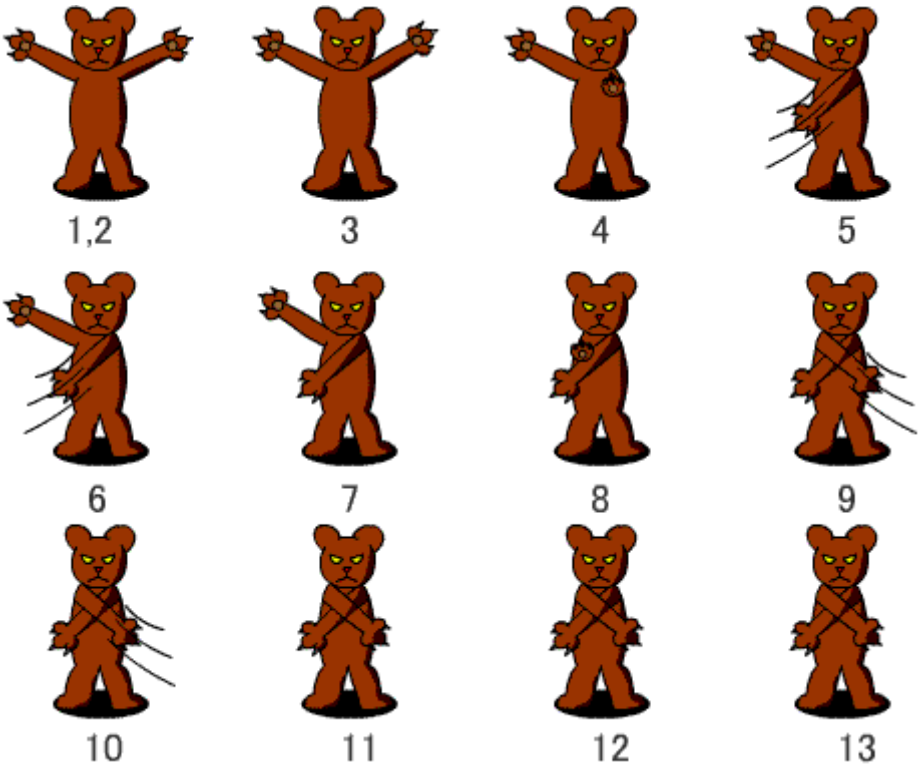


図 4－2－14　モンスター「暴れグマ」の動き(数字はフレーム番号)

7　イベントで出題するクイズを考える

クイズを考える前にまず考えたことは、ただ闇雲にアクションスクリプトの中からクイズを出すのではなく、1 からアクションスクリプトを学習する場合、どういった順序で学習していけばよいのかと考えてみました。そこでアクションスクリプトに関する文献を参考にし、各スクリプトがどのような順序で紹介されているかをまとめて見ました。

(1)ハンドラ

どんな時にそのスクリプトが実行するのかを定義するもの。

(例) on()、onClipEvent()

(2)プロパティ

そのオブジェクトが持つ属性の事を指す。座標や大きさ透明度などを指す。

(例) _x、_y、_visible、_alpha、_currentframe、_root　など

(3)アクション

オブジェクトなどに関係なく、単独で実行できるコマンド。

(例) if、for、function、gotoAndPlay()、trace、など

(4)オブジェクト

①グローバルオブジェクト

キーボード入力やカーソルの表示･非表示などムービー全体で 1 つしか存在しないようなものを管理するオブジェクト。

ムービー再生時に自動的に生成される。Key オブジェクト(キーボードからの入力)や Math オブジェクト(数値演算)などがある

(例)Key.isDown、Math.random() など

②スクリプトで生成するオブジェクト

スクリプト上で使われる目に見えないオブジェクト。アクションスクリプトで扱う、文字列、配列の値などはすべてオブジェクトです。文字列は「String オブジェクト」、配列は「Array オブジェクト」です。これらはスクリプト上でそれを使ったとき自動的に生成されます。

(例)Array.push()、String.length()、Date.getHours()　など

プロパティやアクションは文献によって説明の順番は異なるが、ハンドラについてはほとんどの文献で最初のほうに説明されていることがわかりました。スクリプトに関するクイズを出す場合ははじめのほうでハンドラについて説明し、後のほうでオブジェクトに関するクイズを出すようにしました。

8　マップや建物などのシンボル作成

　次にマップ作成の手順について説明します。

「新規シンボル」でムービークリップを選択し名前を「壁」、「移動イベント」とします。

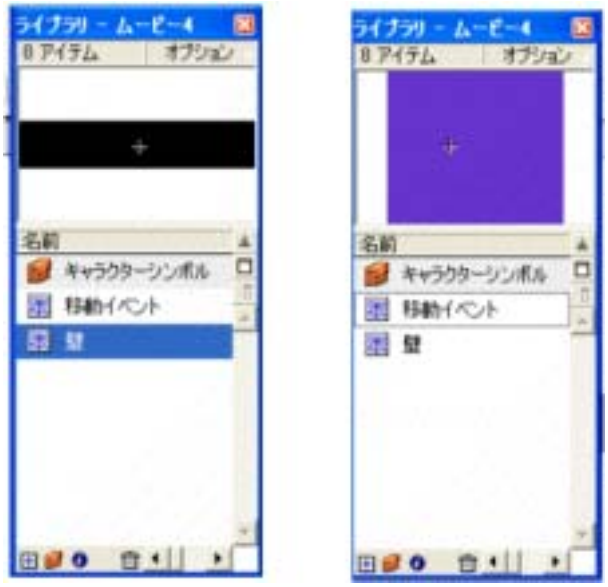


図 4－2－15　ムービークリップ「壁」、「移動イベント」

新規シンボルでムービークリップ名を「マップ 1」とし、「壁」、「移動イベント」のムービークリップを配置します、下の図のように壁を配置します。

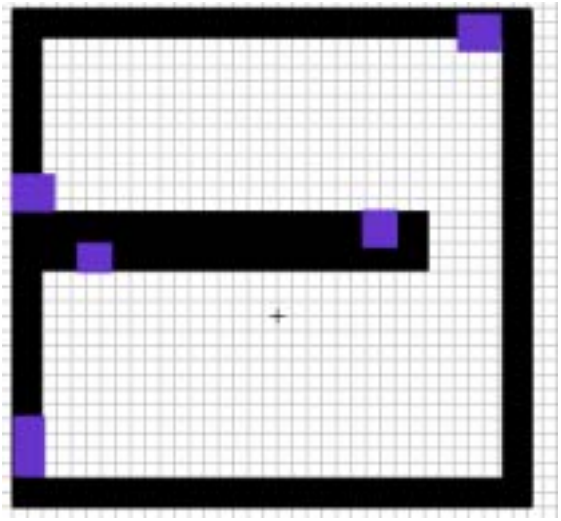


図 4－2－16　マップの壁を作成

次にその上に貼り付けていくアイテムを作ります。

アイテムを貼り付けていく場合、タイムラインの上から順に、レイヤーに配置したオブジェクトが重なり合うように表示されます。

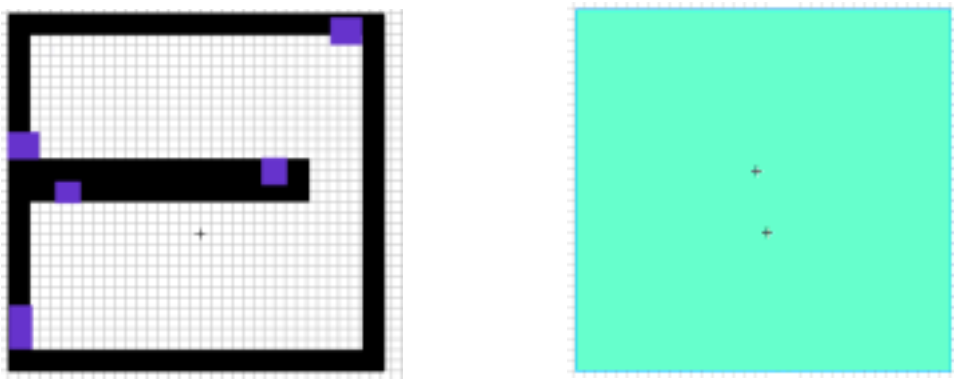


図 4－2－17　レイヤーに配置するオブジェクト

その例を見て見ましょう。図 4-2-17 の左側のマップ部分はレイヤー１の 1 フレーム目に、右側の塗りの部分はレイヤー2 の 1 フレーム目に配置されたオブジェクトです。

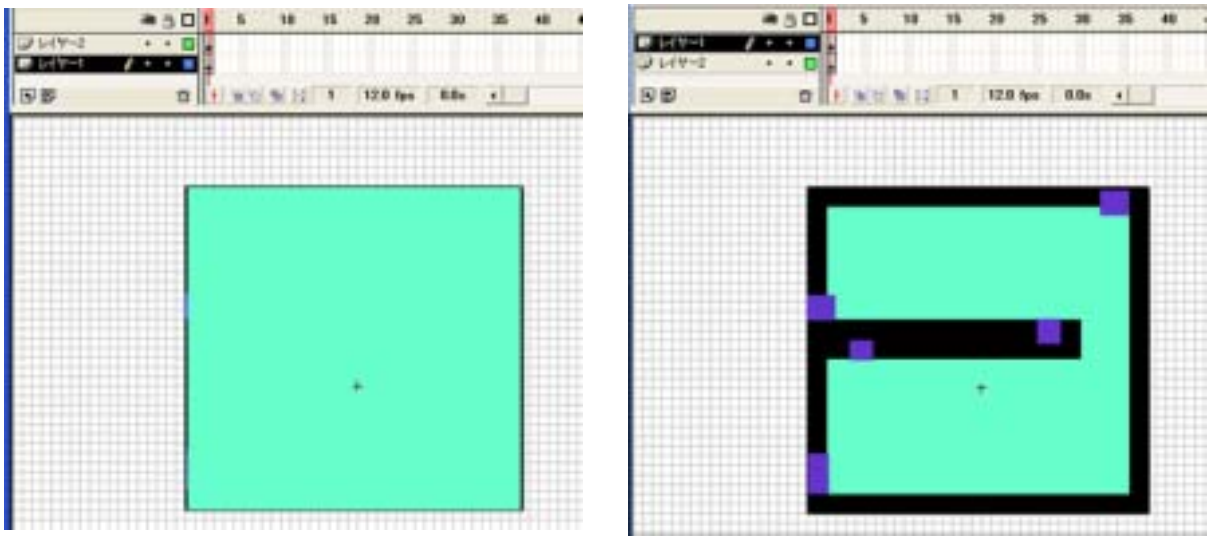
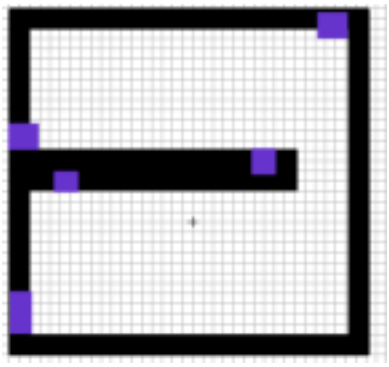


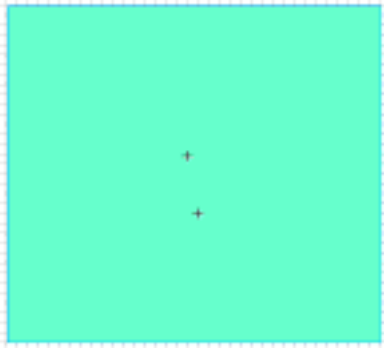
図 4－2－18　ステージで見たオブジェクト

図 4-2-18 の左側はレイヤー1 を下に、右側はレイヤー1 を上にした場合の画像です。左側はレイヤー1 のマップの上にレイヤー2 の四角形の塗りを配置したため、マップが見えなくなってしまいました。

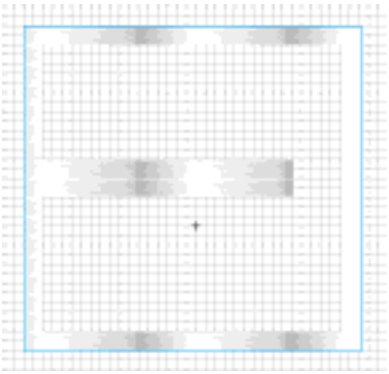
逆に右側は下に四角形の塗りを置き、その上にマップを配置したので、図のようにマップが見える形になりました。
このことからステージ上のオブジェクトはレイヤーの上から順に重なりあうように表示されることがわかります。



レイヤー1



レイヤー2



レイヤー3

レイヤー4

図 4－2－19　レイヤー名と配置するオブジェクト

では、ゲームで使用されるマップを作成していきます。まず、レイヤー1 には「壁」と「移動イベント」を配置したマップを、レイヤー2 には床となるオブジェクトを、レイヤー３は「壁」の上に貼る画像を、レイヤー4 には「移動イベント」の上に貼るオブジェクトを作成しました。それらをタイムラインの上から順に、レイヤー4、レイヤー3、レイヤー2、レイヤー1 と配置していくと図 4-2-20 のようになります。

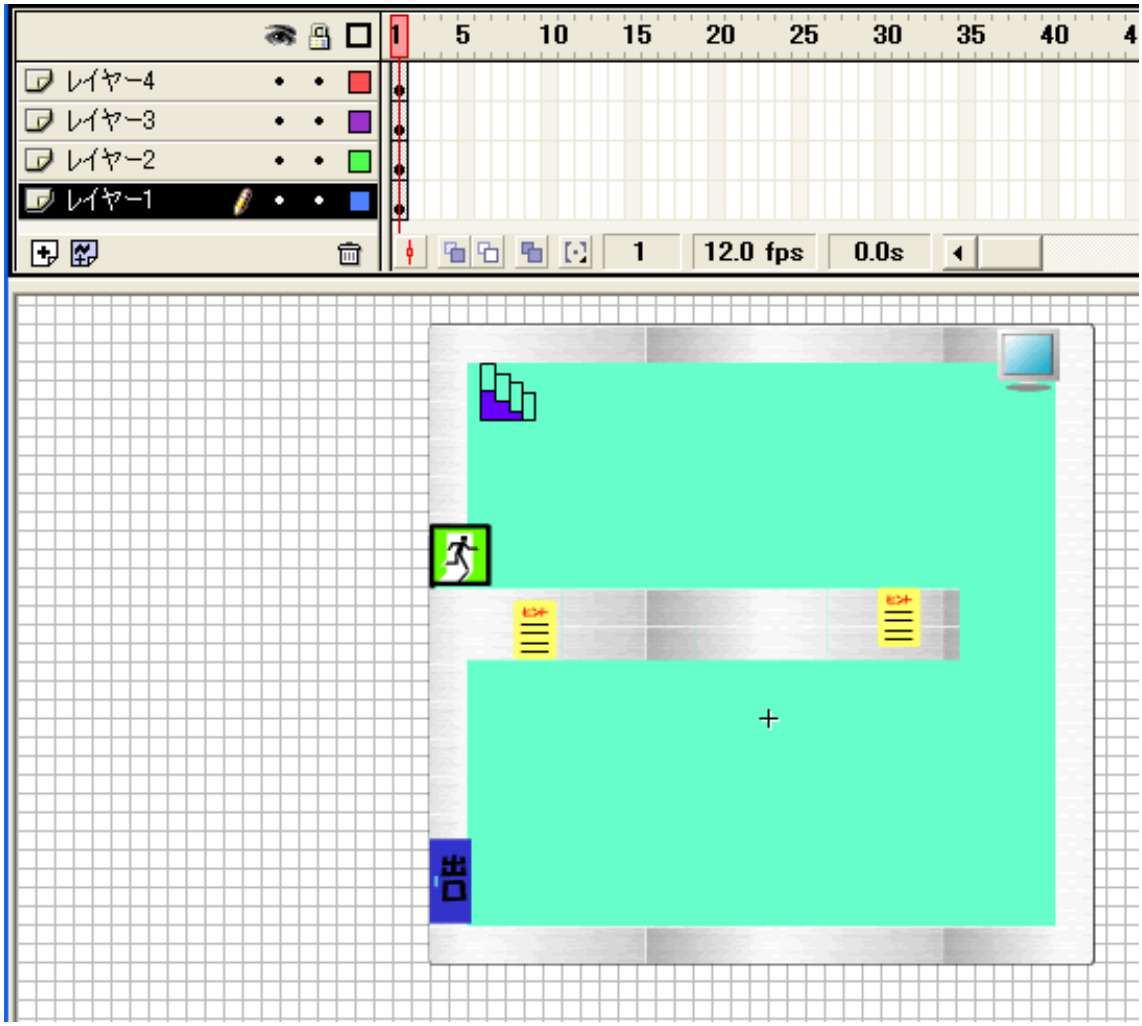


図 4－2－20　出来上がったマップ

これで、ムービークリップ「マップ 1」は完成です。ここで作成した「マップ 1」は第 5 章の 5.1 のマップ移動の際に説明します。

同様にほかのマップもこのように作成していきます。作成したシンボルをステージに配置します。

9　プログラム作成

　シンボルをステージに配置したら次はプログラムを作成します。

ゲームの場面分けると

- タイトルが表示されるまで(1 から 6 フレーム目まで)
- ゲームスタート時(7 から 9 フレーム目まで)
- マップ移動(10 から 15 フレーム目まで)
- ショップなどに行ったとき(18 から 27 フレーム目まで)
- クイズイベント(29 から 32 フレーム目まで)
- 戦闘シーン(34 から 60 フレーム目まで)
- ゲーム終了後のおまけ(61 から 63 まで)

となります。

　各フレームでどういった処理をしているのか説明していきます。

1、2 フレーム

　1 フレーム目であるフレームまでデータを読み込んだら 3 フレーム目に再生ヘッドを移動するようにしている。2 フレーム目には ”gotoAndPlay(1)” と記述してあるので、フレームが読み込まれるまでは、1 フレーム目に戻るようにしている。

3 フレーム

　初期化や関数の定義。ここでは、アイテム、主人公・敵キャラクターの能力値などのデータの配列が記述してある。またアイテム、モンスターの加算などの関数も定義している。

4 フレーム

　コマンドメニューの初期化。メニューを選んだときに表示されるコマンドの配列などが記述してある。主人公の出現位置や能力値を初期化している。

5、6 フレーム

　1､2 フレームと同じように､あるフレームまで読み込まれたら 7 フレーム目(タイトル画面)にいくようにしている。

7 から 9 フレーム

　ボタンを押すと次のフレームにいくようにしている。7 フレーム目で ”eventFlag”を”false”にし､9 フレーム目ではトークテキスト画面にテキストを表示させるようにしている。

10 から 15 フレーム
　フレームでは主人公をマップの中央に配置し、マップを指定の位置に表示するスクリプトが記述されている。マップの移動については、ムービークリップ名「キャラクター」の中にアクションスクリプトが記述されているため、第 5 章で説明する。

18 から 27 フレーム
　ショップや休憩室などに来た時にコマンドやテキスト画面を表示させるようしている。

29 から 32 フレーム
　そのフレームで再生ヘッドを停止させている。クイズイベントのルーチンについてはマップ移動同様に、第 5 章で説明する。

34 フレーム　(ラベル名 ”battle”)
　戦闘シーンの Flag を”true”にする、すなわち、戦闘が始まることを意味する。敵キャラクターの絵を画面に表示し、同時に 2 フレーム目から敵キャラクターの情報も持ってきて、加算する。

35 フレーム　(ラベル名 “AttackMenu”)
　「たたかう」、「どうぐ」、「じょうたい」、「にげる」コマンドを表示。

36 フレーム　(ラベル名 ”AttackLoop”)
　35 フレームと同じスクリプトが記述されている

37 フレーム
　テキスト画面が表示されている時、36 フレーム目の”AttackLoop”に戻る。戦う相手、使用するアイテムを選ばない限り、次のフレームに進めない。

38 フレーム　(ラベル名　“AttackExec”)
　主人公の攻撃ルーチン。敵に与えるダメージ、戦闘が終わった後にもらえる「経験値」「お金」などを計算している。レベルアップに必要な経験値がたまったら主人公のステータスを更新し、”LevelUp”のラベルへ、再生ヘッドを移動。

39 から 46 フレーム　(ラベル名　“monAttackInit”)
　敵キャラクターの攻撃の初期化。

47 フレーム　(ラベル名　“monsterAttack”)
　敵キャラクターが主人公に与えるダメージ計算を行っている。主人公の HP が 0 より小さくなるとラベル名 ”charadown”のフレームに戻るようにする。テキスト画面を表示するようにしている。

48 から 51 フレーム　(ラベル名　“monsterAttackLoop”)
　47 フレームと同じ処理をしている。

52 フレーム
　全部の敵の攻撃が終了しているかいないかを判断し、条件に応じて異なる処理をするようにしている。

53 フレーム　(ラベル名　"LevelUp”)
　レベルアップしたことをテキスト画面に表示する。

55 から 60 フレーム　(ラベル名　“LevelUpLoop”)
　テキスト画面を非表示にすると、戦闘シーンに入る前のフレームへ再生ヘッドを移動する。(マップに戻る)

61 から 63 フレーム
　ボタンを押すまでそのフレームで停止させておく。

10　テストプレイ

　プログラムを組んだ後、実際に自分の思ったとおりに動くかどうか「ムービープレビュー」でテストプレイをします。

テストプレイがうまくいくと、次は実際に Web で公開する場合、ダウンロードの待ち時間がどのくらいなのか調べます。

　まずは「制御」メニューの「ムービープレビュー」を選択します。

図 4－2－21　ムービープレビューで確認

　ムービープレビューを実行し、「表示」メニューの「プロファイラ」を実行すると、図 4-2-22 のように、画面の上のほうにグラフが表示されます。「デバッグ」メニューで通信速度を選ぶことができ、この図では「56 k(4.7KB/s)」を選択しています。グラフの赤い線を越えている箇所ではフレーム再生までにデータが読み込まれていないと、ムービーが中断する原因になります。この図の場合は、4、34 フレームが大きくバーを越えています。

　図 4-2-23 より、4 フレーム目にはゲームで使用するテキスト画面、メニューコマンドなどがここで初めて登場し、またメニューコマンドの中にはスクリプトが多数記述されていることから、データ量が重いのだと考えます。

　34 フレーム目は戦闘シーンの最初の部分を示しており、このフレームでは「モンスター」という 19 の敵キャラクターの動きが入ったムービークリップを読み込んでくるために、データ量が重くなったのだと考えます。

図 4－2－22　プロファイラを実行

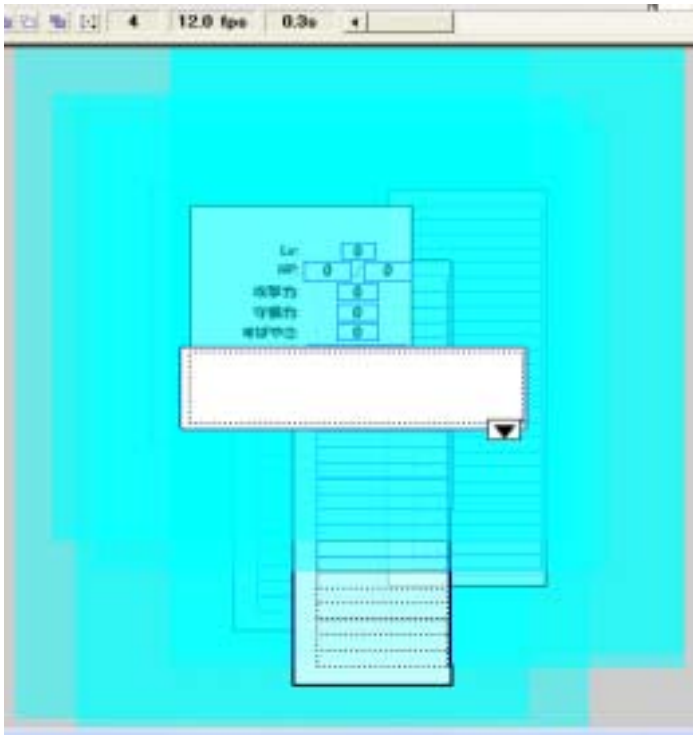


図 4－2－23　フレーム 4 に配置されているオブジェクト

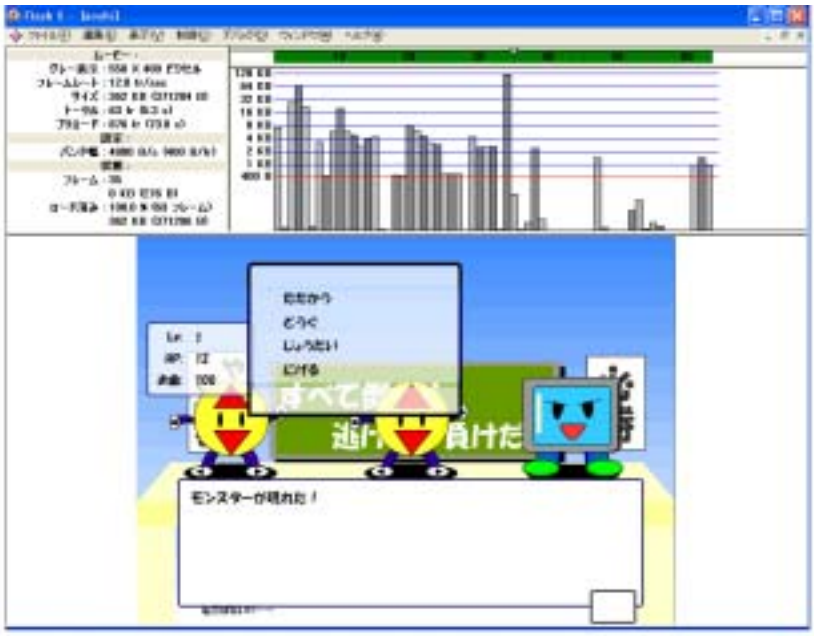


図 4－2－24　戦闘シーン(データが読み込まれている場合)

では、もしデータが読み込まれずに再生されるとどうなるのでしょうか。図 4-2-24 は戦闘シーンのデータが読み込まれて再生した場合を示します。グラフの上の緑のバーが最後のフレーム目まで伸びていることからデータがすべて読み込まれたということがわかります。

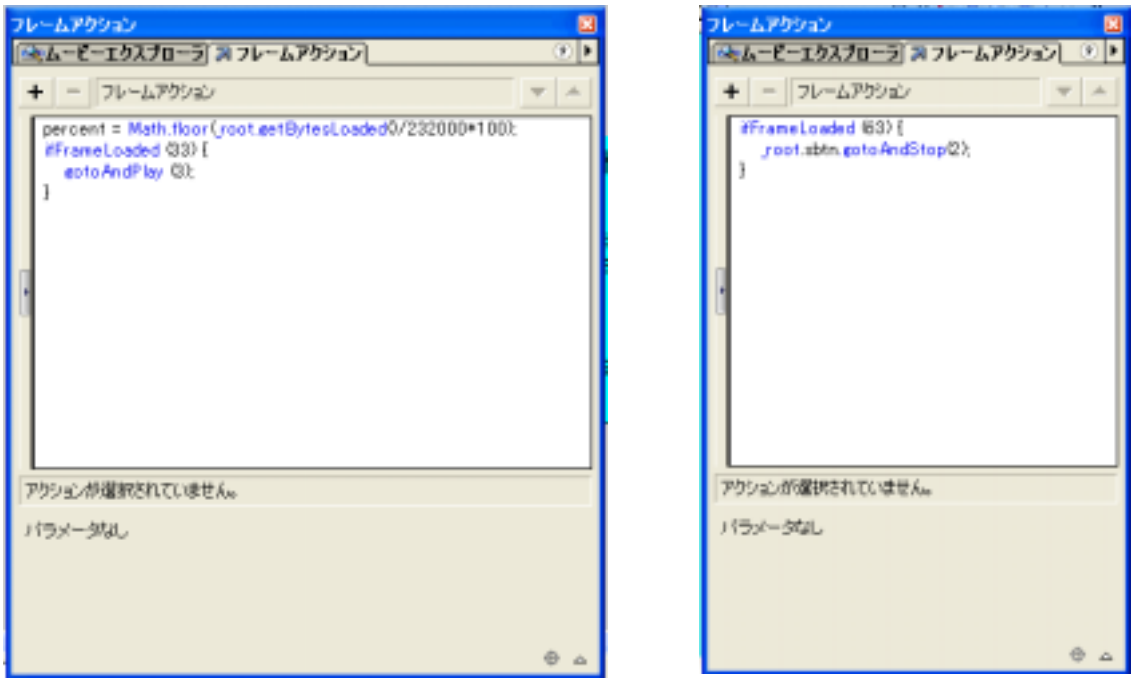


図 4－2－25　フレームアクション(左が 1 フレーム目、右が 5 フレーム目)

図 4-2-25 はシーン 1 の 1 フレーム目と 5 フレーム目のフレームアクションです。1 フレーム目のフレームアクション

```
ifFrameLoad(33){
  gotoAndPlay(3)
}
```

は、”33 フレームが読み込まれたら 3 フレーム目へ移動する。”を意味します。33 フレームは戦闘シーンが始まる前のフレームです。

同様に 5 フレーム目のフレームアクションは

```
  ifFrameLoad(63){
  _root.sbtn.gotoAndStop(2)
}
```

は、”63 フレーム目が読み込まれたらシーン 1 に配置されたインスタンス名「sbtn」の 2 フレームは再生ヘッドを移動し、停止” を意味します。

ここで出てくる ”インスタンス名” とは、ライブラリからステージに配置したムービークリップにつける名前のことを言います。インスタンス名をつけておくと他のムービークリップなどからコントロールすることができます。”sbtn” の 2 フレーム目にはボタンが配置されており、それをクリックするとタイトル画面に再生ヘッドを移動させるようにしています。5 フレーム目ではすべてのデータが読み込まれるまで先に進めないようにしています。

もし 5 フレーム目を削除し、1 フレーム目のアクションが終了するとそのまま”stop();”をせずにタイトル画面にいくようにするとどうなるでしょうか。

1 フレームのアクションが終わった時点で戦闘シーンまでのデータは読み込んでいるのだから、そこまでの分については問題なくゲームをできます。その間に残りのデータを読み込んでくれるのだからなくても問題がないように見えます。しかし、戦闘シーンの読み込みが終了する前に、そのフレームに行ってしまうと図 4-2-26 のように、戦闘シーンに入ったけど敵のグラフィックが画面に表示されず、又図 4-2-24 のようなコマンドも表示されないといった問題が発生します。

図 4-2-27 は 1 フレーム目、5 フレーム目のダウンロードの待ち時間に表示されるオブジェクトを示しています。1 フレーム目ですべてのデータを読み込むまで先に進めないようにすることもできますが、それだと待ち時間が長く感じられるようになるので、戦闘シーンの前まで 1 フレーム目である程度データを読み込んでおいて、5 フレーム目ですべてのデータを読み込むまでに、ゲームの説明や注意を読んでもらったほうがよいのではないかと思い作成しました。

図 4－2－26　戦闘シーン(データが読み込まれていない場合)

図 4－2－27　レイヤー名「背景」に配置してあるオブジェクト
(左が 1 フレーム目、右が 5 フレーム目だけに配置)

第 5 章　主ルーチンの説明

この章ではゲームのメインとなる箇所のプログラムについて説明していきます。

5.1　マップ移動

作成したゲームはキャラクターをステージの中心に配置し、キャラクターはその位置から動かずに、マップを動かすことによってキャラクターがマップ上を歩いているように見せています。

ルーチンの説明に入る前にここで使用される主なスクリプトについて説明します。

・ if　--- 条件が満たされた時、{}の中に記述されたアクションを実行する。

書式：　if(条件){
　　条件が正しい時に実行したい処理
}

・ else --- if 文と一緒に使い、「---でなければ」を意味する。

書式：　if(条件){
　　条件が正しい時に実行したい処理
}else{
　　条件が正しくない時に実行したい処理
}

・ else if --- if 文と一緒に使い、「---ではなく、もし---ならば」を意味する。

書式：　if(条件 1){
　　条件 1 が正しい時に実行したい処理
}else if(条件 2){
　　条件 1 が正しくなく、
　　条件 2 が正しい時に実行したい処理
}

・ for --- 決まった回数、繰り返して処理をする。
書式： for(初期値；条件；増分){
繰り返す処理
}

・ hitTest --- インスタンス同士の衝突を判定する。
書式： インスタンス名.hitTest(対象インスタンス)

・ Key.isDown --- 指定したキーが押されているかどうか調べる
書式： Key.isDown(キーコード)

・ Math.floor --- 小数点以下を切り捨てて整数化する。
書式： Math.floor(数値)

・ Math.random --- 0 以上 1 未満の乱数を発生する。
書式： Math.random()

プログラムを作成する前に作成したシンボルにインスタンス名をつけます。
まずはじめに、ムービークリップ「壁」のインスタンス名を「w1」「w2」---「w12」とします。ムービークリップ「移動イベント」のインスタンス名をそれぞれ、「exit」「hint」「hint1」「pc」「out」とします。

図 5－1－1　ムービークリップ「壁」「移動イベント」のインスタンス名

「壁」「移動イベント」を配置したムービークリップ「マップ 1」をステージに配置しインスタンス名を「map1」とします。(図 5-1-2)
ムービークリップ「マイキャクター」のインスタンス名を「chara」とします。この「マイキャラクター」は図 4-2-12 の前後左右の動きを貼り付けたムービークリップを使います。
この「マイキャラクター」は「マップ 1」の上のレイヤーに配置します。
ラベル名「マイキャクター」にはムービークリップ「マイキャラクター」を、マップにはムービークリップ「マップ 1」を配置します。

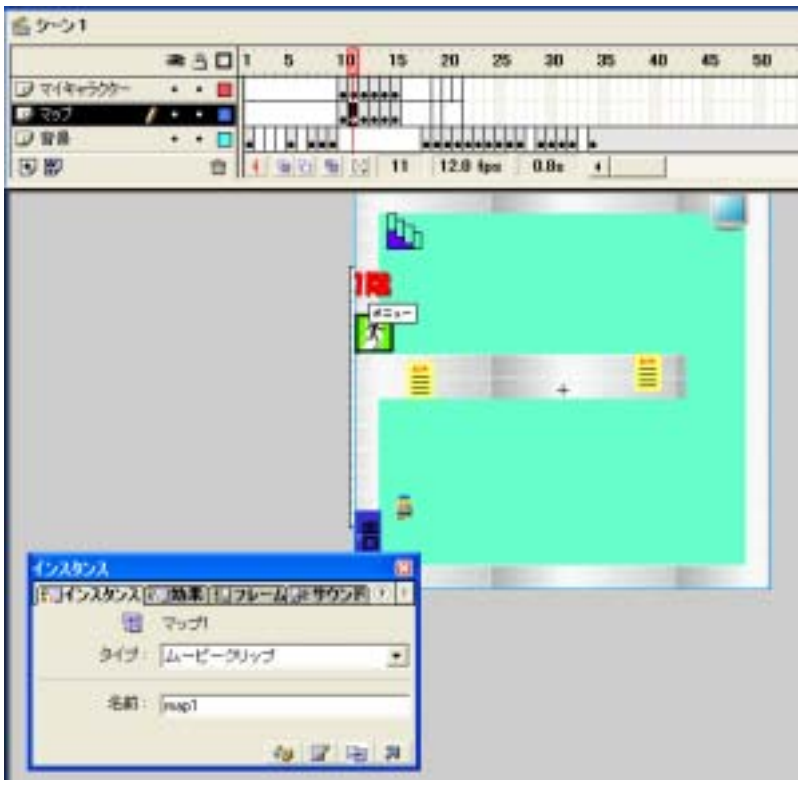


図 5－1－2　「マップ 1」のインスタンス名

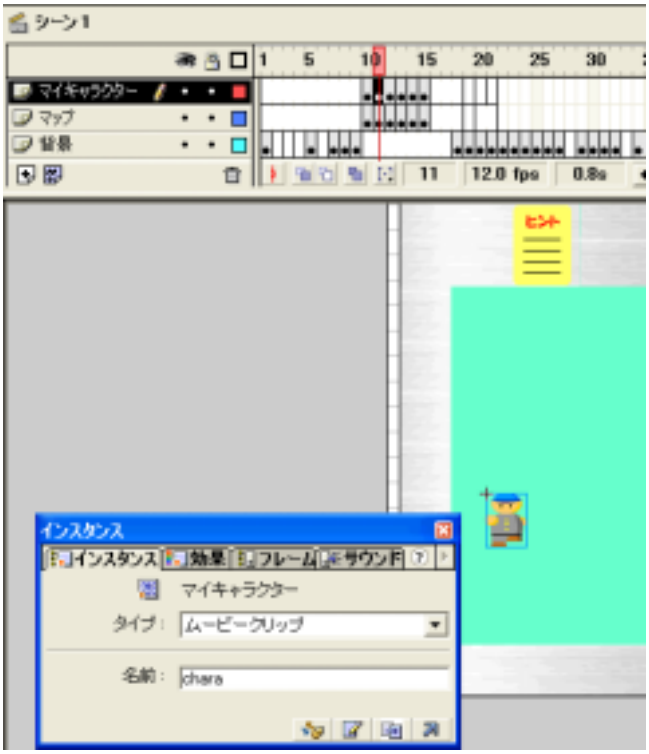


図 5－1－3　「マイキャラクター」のインスタンス名

シンボルの配置がすんだら、次はプログラムの作成です。
「アクション」レイヤーの 4 フレーム目のフレームアクション

```
centerX = 275;
centerY = 200;
myX = -50;
myY = -150;
MoveDist = 18;
```

は中心点として「centerX」「centerY」に「275」「200」と記憶します。
これはムービーのサイズが 550 × 400 なのでそのように設定しました。
最初に地図上に現れる位置として「myX = －50」「myY = －150」と記憶しておきます。「MoveDist=18」は移動量を表します。

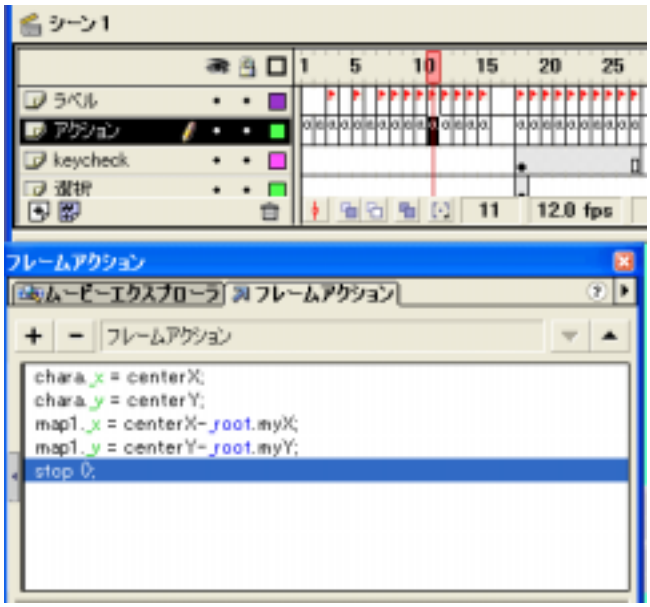


図 5－1－4　「アクション」レイヤーの 11 フレーム目のフレームアクション

11 フレーム目のフレームアクションは以下のようになります。

```
chara._x = centerX;                 // chara の x 座標を centerX(=275)にする
chara._y = centerY;                 // chara の y 座標を centerY(=200)にする
map1._x = centerX-_root.myX;        // map1 の x 座標をステージ中央から自分の
                                    位置だけ引いた場所にする(275－(－50))
map1._y = centerY-_root.myY;        // map1 の y 座標をステージ中央から自分の
                                    位置だけ引いた場所にする(200－(－150))

stop ();                            //  このフレームで停止する
```

11 フレーム目のムービークリップ「マイキャラクター」のアクションスクリプトは以下のようになります。

```
onClipEvent (enterFrame) {  // ムービークリップが再生されているとき
        var vx = 0;  // 変数 vx(横の移動量)に 0 を入れる
        var vy = 0;  // 変数 vy(縦の移動量)に 0 を入れる
        if (_root.moveFlag eq false) {  // moveFlag が false の時
                return; // それ以降のスクリプトは実行しない
        }
        if (Key.isDown(Key.RIGHT)) {  // 「→」キーが押されていたら
                _root.chara.gotoAndStop(3);  // chara の 3 フレーム目へ移動し、
                vx = _root.MoveDist;  // vx に MoveDist の値(=18)を入れる
        } else if (Key.isDown(Key.LEFT)) {  // 「←」キーが押されていたら
                _root.chara.gotoAndStop(4);  // chara の 4 フレーム目へ移動し、
                vx = -_root.MoveDist;  // vx に－MoveDist の値を入れる
        } else if (Key.isDown(Key.DOWN)) {  // 「↓」キーが押されていたら
                _root.chara.gotoAndStop(1);  // chara の 1 フレーム目に移動し、
                vy = _root.MoveDist;  // vy に MoveDist の値を入れる
        } else if (Key.isDown(Key.UP)) {  // 「↑」キーが押されていたら
                _root.chara.gotoAndStop(2);  // chara の 2 フレーム目に移動し、
                vy = -_root.MoveDist;  // vy に－MoveDist の値を入れる
        }
        _root.map1._x -= vx;  // map1 の x 座標から vx だけ引く
        _root.map1._y -= vy;  // map1 の y 座標から vy だけ引く
        var clashFlag = false;  // 衝突判定フラグ clashFlag を false にする
        for (i=1; i<=12; i++) {  // ムービークリップ「壁」の数だけ繰り返す
        if (this.hitTest("_root.map1.w"+i)) {  // map1 の壁と衝突していたら
                        clashFlag = true;  // clashFlag を true にする
                }
        }
        //
        if (this.hitTest(_root.map1.next1)) {  // 「next1」に衝突し、
                if (_root.event1Flag == false) {  // event1Flag が false ならば
                        _root.gotoAndStop("quiz1");  // ラベル名「quiz1」のフレームに移動する
```

```
                } else {    // そうでなければ(event1Flag が true ならば)
                _root.myX = -230;    // 次のマップでの出現位置 myX に－230 を
                        _root.myY = -250;    // myY に－250 を入れ、
                        _root.gotoAndStop("map2");    // ラベル名「map2」のフレームへ移動
                }
        }
        if (this.hitTest(_root.map1.out)) {    // 「out」に衝突したら
                _root.myX = -50;    // myX に－50 を入れ、
                _root.myY = 40;     // myY に 40 を入れ、
                _root.gotoAndStop("zentai");    // ラベル名「zentai」のフレームへ移動
        }
        if (this.hitTest(_root.map1.exit)) {    // 「exit」に衝突したら
                _root.myX = -50;    // myX に－50 を入れ、
                _root.myY = 40;     // myY に 40 を入れ、
                _root.gotoAndStop("zentai");    // ラベル名「zentai」のフレームへ移動
        }
        if (this.hitTest(_root.map1.pc)) {    // 「pc」に衝突したら
                _root.returnframe = _root._currentframe;
// returnframe に現在のフレーム番号を入れ
                _root.myX = _root.centerX-_root.map1._x-vx;
// 衝突する前の自分の位置を myX に入れておき、店などから戻ってきた時に出現する位置を覚えておく
                _root.myY = _root.centerY-_root.map1._y-vy;
// 衝突する前の自分の位置を myY に入れておき、店などから戻ってきた時に出現する位置を覚えておく

                _root.gotoAndStop("netpc1");    // ラベル名「netshop1」のフレームへ移動
        }
        if (this.hitTest(_root.map1.hint)) {    // 「hint」に衝突したら、
                _root.DispTalk.TalkText = String.fromCharCode(13)+"    「どうぐ」コマンドで、SAN-Q や P☆Bag で買ったアイテムが見れます。
"+String.fromCharCode(13);    // 「DispTalk」(インスタンス名)の中の、
```

「TalkText」(変数)に ” ”の中のテキストを表示させる
_root.DispTalk.TalkText +=" 「ちしき」コマンドで、ネットショップで買った情報が見れます。"+String.fromCharCode(13);
_root.DispTalk._visible = true; // 「DispTalk」を表示させる
_root.moveFlag = false; // moveFlag を false にし、
clashFlag = true; // claseFlag を true にする
}
if (this.hitTest(_root.map1.hint1)) { // 「hint1」に衝突したら、
_root.DispTalk.TalkText = String.fromCharCode(13)+"非常口を使うと建物の外に出られます。"+String.fromCharCode(13);
_root.DispTalk.TalkText += "ネットショップで情報を買っておくと、この先のクイズイベントで有利になる"+String.fromCharCode(13);
_root.DispTalk.TalkText += "・・かも。"; // 「DispTalk」の中の、「TalkText」に ” ”の中のテキストを表示させる
_root.DispTalk._visible = true; // 「DispTalk」を表示させる(DispTalk の_visible を true にする)
_root.moveFlag = false; // moveFlag を false にし、
clashFlag = true; // claseFlag を true にする
}
//
if (clashFlag eq true) { // clashFlag が true の時(衝突していたら)
_root.map1._x += vx; // map1 の x 座標に vx を加える
_root.map1._y += vy; // map1 の y 座標に vy を加える
} else if (vx+vy ne 0) { // そうでなくもし、移動量が 0 でなければ
_root.MeetMonster--; 「MeetMonster」(変数)の値を減らす(1 歩ごとに MeetMonster の値を 1 ずつ減らしていく)
if (_root.MeetMonster<0) { // もし「MeetMonster」の値が 0 より小さくなれば
_root.MeetMonster = 14+Math.floor(Math.random()*4); // 「MeetMonster」の値を新たに設定する
_root.returnframe = _root._currentframe;
// returnframe に現在のフレーム番号を入れ
_root.myX = _root.centerX-_root.map1._x-vx;
// 衝突する前の自分の位置を myX に入れておき、戻ってきた時に出現する位置を覚えておく

```
                    _root.myY = _root.centerY-_root.map1._y-vy;
// 衝突する前の自分の位置を myY に入れておき、戻ってきた時に出現する位置を覚えておく
                    if (340<_root.map1._y) {
// もし map1 の y 座標が 340 以上の時
                        _root.monsterArea = 2;
// 「monsterArea」を 2 にする
                    } else {
// そうでない場合は
                        _root.monsterArea = 1;
// 「monsterArea」を 1 にする
                    }
                    _root.gotoAndPlay("battle");
// 「monsterArea」を設定した後、ラベル名「battle」のフレームへ移動
                }
            }
}
```

図 5－1－5　ゲーム中のマップ移動

5.2　クイズイベント

次にクイズイベントのルーチンについて説明します。
クイズイベントで必要となるオブジェクトは以下のとおりです。

ボタン名　「クイズイベント透明ボタン」
　正解だと「クイズイベント」の次のフレームに進むようにしている。

ムービークリップ名　「MovieClip 選択」
　1 フレーム目に「qbtn 答え 1」,「qbtn 答え 2」が配置されている。2 フレーム目に正解した時の、3 フレーム目に不正解の時の絵が描いてある。
　インスタンス名：sentaku

ボタン名　「qbtn 答え 1」 、「qbtn 答え 2」
　ボタンをクリックすると、値を返すようにしている。 「MovieClip 選択」の中に配置されている。

ムービークリップ名　「クイズイベント」
　2、3、4 フレーム目で Math.random()を使用して問題を選び、「MovieClip 選択」の 1 フレーム目で停止している。
　インスタンス名：question1

ムービークリップ名　「Talk テキスト 2」
　コメントを表示する。
　インスタンス名：DispTalk2

出題するクイズはシーン 1 の 3 フレームの quizData で配列を定義しています。

```
quizData = new Array();　// 「quizData」という空の配列を用意する
for (i=0; i<=28; i++) {　// i が 0 以上 28 以下の場合、以下の処理を繰り返す
　　quizData[i] = new Object();　//オブジェクトを生成
}
quizData[0].Que = "Q. ボタンアクションのイベントに使用されないのは？
　"+String.fromCharCode(13);　// quizData[0]のクイズ問題
quizData[0].Ans1 = String.fromCharCode(13)+" 1.　release";
// quizData[0]の解答 1
quizData[0].Ans2 = String.fromCharCode(13)+" 2.　enterFrame";
// quizData[0]の解答２
```

```
quizData[0].seikai = 2;   // quizData[0]の正解
```

各オブジェクトのアクションスクリプトについて説明していきます。

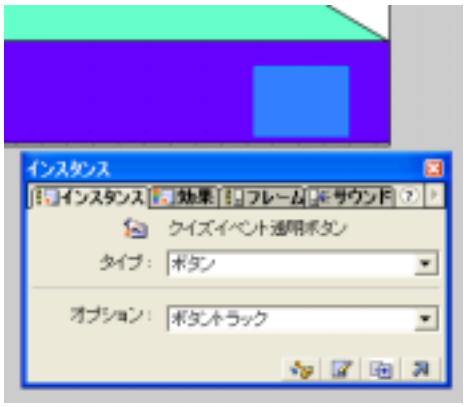


図 5－2－1　ボタン「透明ボタン」

「透明ボタン」のオブジェクトアクションは次のようになります。

```
on (release) {   // ボタンがクリックされると
    _root.DispTalk2._visible = false;   // DispTalk2 を非表示にする
    _root.sentaku._visible = false;   // sentaku を非表示にする
    if (_root.seikai2 == 1) {   // seikai2 が 1 の時(正解の場合)
        _root.question1.nextFrame();   question1 の次のフレームへ移動
        if (_root.question1._currentframe == 5) {
// もし、question1 の 5 フレーム目に来た時
            _root.event1Flag = true;   // event1Flag を true にする
            _root.myX = -230;   // myX に－230 を入れる
            _root.myY = -250;   // myY に－250 を入れる
            _root.gotoAndStop("map2");
// ラベル名「map2」のフレームへ移動
        }
    } else {   そうでなければ(不正解の場合)
        _root.myX = 230;    // myX に－230 を入れる
        _root.myY = -100;   // myY に－100 を入れる
        _root.gotoAndStop("map1");   // ラベル名「map1」のフレームへ移動
    }
}
```

「透明ボタン」ではクリックした時に seikai2 の値をみて、”seikai2 = 1”ならば次のフレームに移動し、”seikai2 = 0”ならば「マップ 1」に移動するようにしています。question1 の 5 フレーム目に来るとそれはクイズに 3 問連続で正解したことを意味し、「マップ 2」へ移動し、”event1Flag = true”とします。

図 5－2－2　ムービークリップ「MovieClip 選択」

「MovieClip 選択」 のオブジェクトアクションは次のようになります。

```
onClipEvent (load) {   // タイムライン上に始めて登場した瞬間
    _visible = false;   // このオブジェクトを非表示にする
    _root.seikai2 = 1;   // 変数 seikai2 に 1 を入れる
}
```

変数 ”seikai2 = 1”はクイズで選んだ解答が正しいことを意味しています。Load された時、seikai2 が 1 になっているのは、クイズイベントの 1 フレーム目には問題は表示されず、透明ボタンを押すことでクイズが出題される 2 フレーム目に移動することができるからです。

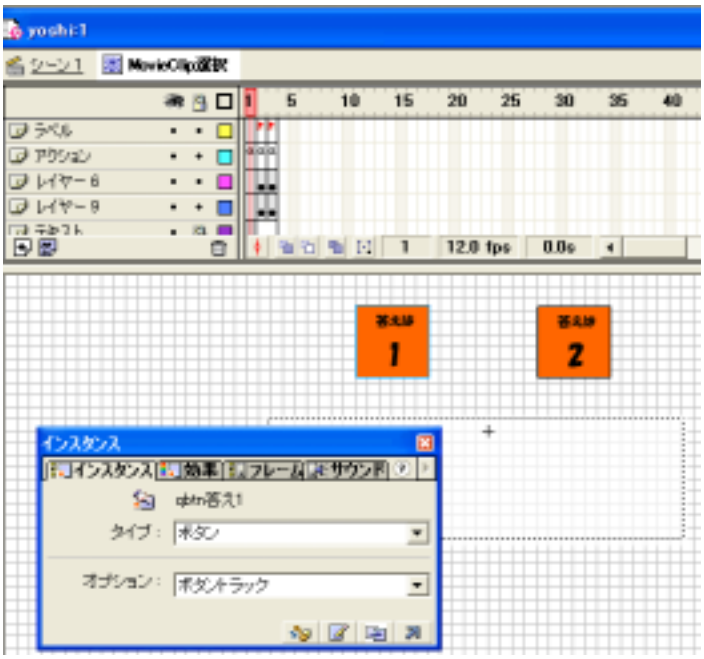


図 5－2－3　ボタン「qbtn 答え 1」

MovieClip 選択の中のフレームアクションは 1、2、3 フレームにはすべて”stop()”が記述されています。2 フレーム目には「正解」の文字、3 フレーム目には「残念」の文字が表示されるようになっています。

1 フレーム目に配置されている、「qbtn 答え 1」のオブジェクトアクションは次のようになります。

```
on (release) {   // ボタンをクリックし
    if (this.seikai == 1) {   // もし seikai が 1 ならば
        _root.seikai2 = 1;   // seikai2 に 1 をいれ、
        gotoAndStop ("atari");   // ラベル名「atari」のフレームへ移動する
    } else {   // そうでなければ
        _root.seikai2 = 0;   // seikai2 に 0 をいれ
        gotoAndStop ("hazure");   // ラベル名「hazure」のフレームへ移動
    }
}
```

ここで出てくる「seikai == 1」とはクイズイベントの 2、3、4 フレームで出題される問題の解答、すなわち、

quizData[i].seikai = 1

の時に「qbtn 答え 1」をクリックしたらクイズに正解したということを意味します。

「qbtn 答え 2」のオブジェクトアクションは次のようになります。

```
on (release) {   // ボタンをクリックし
    if (this.seikai == 2) {   // もし seikai が 2 ならば
        _root.seikai2 = 1;   // seikai2 に 1 を入れ
        gotoAndStop ("atari");   // ラベル名「atari」のフレームへ移動する
    } else {   // そうでなければ
        _root.seikai2 = 0;    // seikai2 に 0 をいれ
        gotoAndStop ("hazure");   // ラベル名「hazure」のフレームへ移動
    }
}
```

「qbtn 答え 2」も「qbtn 答え 1」とほとんど同じスクリプトです。

(例)

```
quizData[2].Que = "Q. クリップアクションの定義の記述として正しいのは？
"+String.fromCharCode(13);
quizData[2].Ans1 = String.fromCharCode(13)+" 1.  on( )";
quizData[2].Ans2 = String.fromCharCode(13)+" 2.  onClipEvent( )";
quizData[2].seikai = 2;
```

上の例のスクリプトが読み込まれた場合、テキスト画面には

Q　クリップアクションの定義の記述として正しいのは？
1．on()
2．onClipEvent()

と表示されます。
この問題では正解は 2 番なので、”seikai = 2” となっています。
この問題が出題された時、もし「qbtn 答え 2」をクリックすれば正解である「atari」のフレームに移動し、逆に「qbtn 答え 1」をクリックすると、不正解である「hazure」のフレームへ移動します。

2、3 フレーム目では、DispTalk2 の_visible を true に、コメントを表示するようにしています。

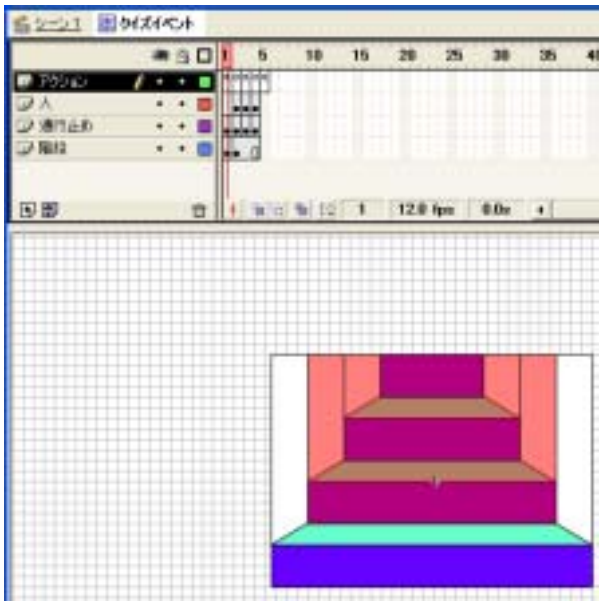


図 5－2－4「クイズイベント」のシンボル編集画面

ムービークリップ「クイズイベント」には 1 から 5 フレームにフレームアクションが記述されています。

1 フレーム目のアクションスクリプトは、

```
_root.DispTalk2.TalkText2 = "これからアクションスクリプトに関するクイズを出題する。"+String.fromCharCode(13);
_root.DispTalk2.TalkText2 += "3 問連続で正解しないと 2 階にはいけない。"+String.fromCharCode(13);  // DispTalk2 に” ”の中のテキストを表示
_root.DispTalk2._visible = true;  // DispTalk2 を表示する
stop ();  // このフレームで停止
```

となり、ここではテキストを表示しているだけです。

2 フレーム目は、

```
quiz1rnd1 = Math.floor(Math.random()*2);  // quiz1rnd1 に 0 か 1 の整数を入れる
_root.sentaku.QuizText= _root.quizData[quiz1rnd1].Que+_root.quizData[quiz1rnd1].Ans1+_root.quizData[quiz1rnd1].Ans2+String.fromCharCode(13);  // quizData[quiz1rnd1]の Que、Ans1、Ans2 の内容を表示
_root.sentaku.seikai = _root.quizData[quiz1rnd1].seikai;  // sentaku の中の変数 seikai に quizData[quiz1rnd1].seikai の値を入れる
```

```
_root.DispTalk2._visible = false;  // DispTalk2 を非表示にする
_root.sentaku._visible = true;  // sentaku を表示
stop ();  // このフレームで停止
```

となります。
このフレームではまずはじめに quiz1rnd1 で値を求めています。
もし quiz1rnd1 = 1 の場合、2 フレーム目の quizData[i]の i に 1 を入れ、

```
quizData[1].Que = "Q. ボタンアクションのイベント 「release」 とは？
"+String.fromCharCode(13);
quizData[1].Ans1 = String.fromCharCode(13)+" 1.  キーボードで指定されたキーが押された時";
quizData[1].Ans2 = String.fromCharCode(13)+" 2.  ボタン上でマウスをクリックした時";
```

を読み込んできて、sentaku の中に配置されてあるダイナミックテキストの変数名 QuizText に表示します。同時に、

```
quizData[1].seikai = 2;
```

も読み込んできて、sentaku の seikai を 2 にします。

3 フレーム目は

```
quiz1rnd2 = Math.floor(Math.random()*3)+2;  //2 以上 4 以下の整数をいれる
_root.sentaku.QuizText=
_root.quizData[quiz1rnd2].Que+_root.quizData[quiz1rnd2].Ans1+_root.quizData[quiz1rnd2].Ans2+String.fromCharCode(13);
// quizData[quiz1rnd2]の Que、Ans1、Ans2 の内容を表示
_root.sentaku.seikai = _root.quizData[quiz1rnd2].seikai;
// sentaku の中の変数 seikai に quizData[quiz1rnd2].seikai の値を入れる
_root.DispTalk2._visible = false;  // DispTalk2 を非表示にする
_root.sentaku.gotoAndStop(1);  // sentaku の 2 フレーム目で停止
_root.sentaku._visible = true;  // sentaku を表示
stop ();  // このフレームで停止
```

となります。quiz1rnd2 に 2、3、4 のいずれかの値をいれて、quizData[i]の i に quiz1rnd2 の値を入れ、それらの配列を読み込んできて sentaku の QuizText に内容を入れます。「クイズイベント」の 3 フレームに来る前には sentaku の中の 2 フレーム目(“正解”という文字が表示されているフレーム)にいるので 1 フレーム目に戻って、再び「qbtn 答え 1」、「qbtn 答え 2」を表示した状態にします。

4 フレーム目は、

```
quiz1rnd3 = Math.floor(Math.random()*3)+5;   // 5 以上 7 以下の整数を入れる
_root.sentaku.QuizText=
_root.quizData[quiz1rnd3].Que+_root.quizData[quiz1rnd3].Ans1+_root.quiz
Data[quiz1rnd3].Ans2+String.fromCharCode(13);
// quizData[quiz1rnd3]の Que、Ans1、Ans2 の内容を表示
_root.sentaku.seikai = _root.quizData[quiz1rnd3].seikai;
// sentaku の中の変数 seikai に quizData[quiz1rnd3].seikai の値を入れる
_root.DispTalk2._visible = false;     // DispTalk2 を非表示にする
_root.sentaku.gotoAndStop(1);     // sentaku の 2 フレーム目で停止
_root.sentaku._visible = true;     // sentaku を表示
stop ();   // このフレームで停止
```

となり、3 フレームとほとんど同じアクションになります。

5 フレーム目には”stop()”アクションが記述されているだけなので、5 フレーム目に来た瞬間、「透明ボタン」の以下のアクションを実行し「map2」のフレームへ移動します。

```
        if (_root.question1._currentframe == 5) {
            _root.event1Flag = true;
            _root.myX = -230;
            _root.myY = -250;
            _root.gotoAndStop("map2");
        }
```

ムービークリップ「Talk テキスト 2」のオブジェクトアクションは、

```
onClipEvent (load) {   // タイムライン上に始めて登場した瞬間
    _visible = false;   // このオブジェクトを非表示にする
}
```

となり、「MovieClip 選択」の 2、3 フレーム目で表示させ、「透明ボタン」や「クイズイベント」などで、非表示にしています。

1 階の階段に来た時に発生するクイズイベントの主な流れは次のようになります。

(1) 「クイズイベント」の 1 フレーム目で停止する。この時点では event1Flag は false の状態になっている。「透明ボタン」をクリックすると「クイズイベント」の 2 フレーム目に移動する。

(2) クイズを出題し、「MovieClip 選択」の「qbtn 答え 1」「qbtn 答え 2」のどちらかをクリック(正解だと思うほうのボタンをクリック)

(3) 正解だと “seikai2 = 1”、不正解だと “seikai2 = 0” とする。

(4) 「透明ボタン」をクリックした時、”seikai2 = 0” の場合「マップ 1」のある地点に飛ばされ、 “seikai2 = 1” の場合「クイズイベント」の次のフレームへ移動する。

(5) (2)の繰り返し

(6) 3 問連続で正解すると event1Flag を true にして、「マップ 2」のフレームへ移動する。

(7) 再び 1 階の階段を訪れた時、event1Flag が true だと、クイズイベントは発生しない。event1Flag が false(3 問連続で正解していない場合)だと、(1)からやり直し。

5.3　メニュー

　ここではムービークリップ「メニュー」のルーチンを説明していきます。ムービークリップ「メニュー」はオブジェクトの階層が深く、上の階層から「メニュー」、「メニューアイテム」、「メニューアイテムボタン」となっています。

図 5－3－1　ボタン名「メニューアイテムボタン」

「メニューアイテムボタン」はマウスが上に来ると(オーバー)、色が変わるようになっています。図 5－3－1 より、ダイナミックテキストの変数名を「MenuItemName」とします。

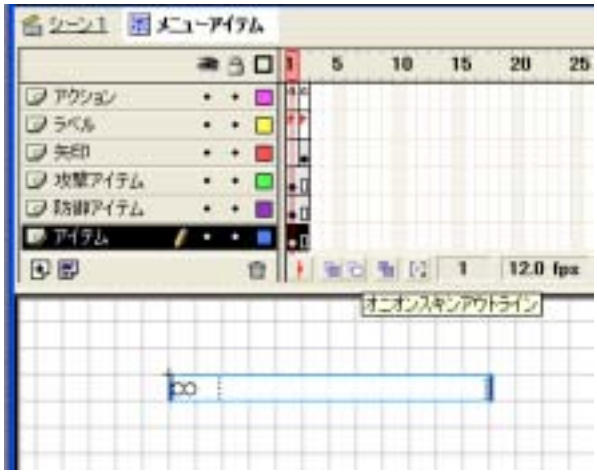


図 5－3－2　ムービークリップ名「メニューアイテム」

「メニューアイテム」は「メニューアイテムボタン」を配置しており、その上のレイヤーに矢印や装備マークとなるムービークリップを配置しています。1、2 フレーム目のアクションは”stop()”が記述されています。

「メニューアイテム」の中の「メニューアイテムボタン」のアクションスクリプトは、

```
on (release) {   // ボタンをクリックすると
    for (name in _parent) {   // _parent に変数 name を代入し、
        eval("_parent."+name).gotoAndStop("markoff");   // _parent の name の「markoff」のフレームへ移動し、停止する
    }
    this.gotoAndPlay("markon");   //  「markon」のフレームに移動する
    _parent.MenuValue = this._name;   // _parent の MenuValue に、this のインスタンス名を入れる
    _parent.gotoAndPlay("MenuExec");   // _parent の「MenuExec」のフレームに移動する
}
```

ボタンアクションの”this”はボタン自身ではなく、それが配置されたタイムライン、すなわち「メニューアイテム」を指します。ここでの ”_parent”はムービークリップ「メニュー」を指します。

「メニュー」の中に配置された「メニューアイテム」はインスタンス名を「0」「1」「2」---「24」とつけているので、eval を用いて文字列で変数にアクセスしています。「メニュー」の MenuValue には「メニューアイテム」のインスタンス名が入り、「メニュー」の「MenuExec」のフレームに移動します。

図 5－3－3　メニューアイテムのインスタンス名

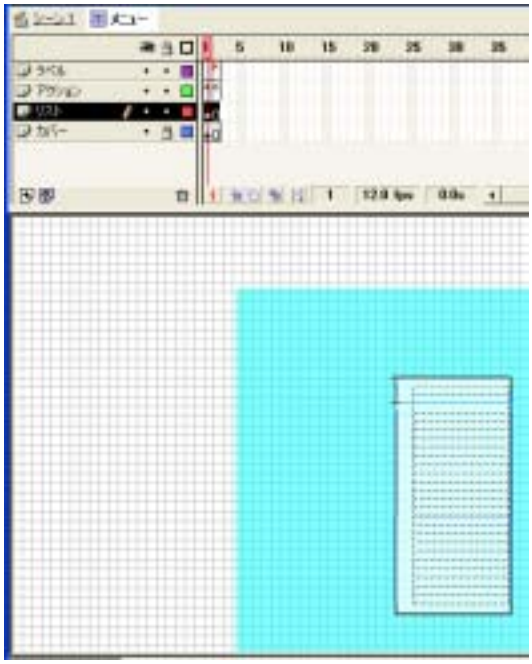


図 5−3−4　ムービークリップ「メニュー」

「メニュー」の中には「メニューアイテム」を複数配置しており、インスタンス名は「0」「1」「2」---「24」としています。「メニューアイテム」のインスタンス名 24 の下には、ムービークリップ「menubottom」を配置し、インスタンス名は MenuBottom としています。「メニュー」の中には 1、2 フレーム目にアクションスクリプトが記述されています。

「メニュー」の 1 フレーム目のアクションスクリプトは、次のようになります。

```
stop ();
function MenuDisp () {    // MenuDisp という関数を定義
      this.MenuNumber = -1;
      DispNum = eval("_root."+cmdName).length;
      for (i=0; i<DispNum; i++) {
            eval(this+"."+i).MenuItemName= eval("_root."+cmdName)[i].Name;
            eval(this+"."+i).MarkArmAtk.gotoAndStop(1);
            eval(this+"."+i).MarkArmDef.gotoAndStop(1);
      }
      resize();
```

```
}
function MenuDisp2 () {    // MenuDisp2 という関数を定義
    DispNum = eval("_root."+cmdName).length;
    for (i=0; i<DispNum; i++) {
        eval(this+"."+i).MenuItemName= eval("_root."+cmdName)[i].Name;
        if (_root.item[i].Arm1) {
            _root.menu2[i].MarkArmAtk.gotoAndStop(2);
        } else {
            _root.menu2[i].MarkArmAtk.gotoAndStop(1);
        }
        if (_root.item[i].Arm2) {
            _root.menu2[i].MarkArmDef.gotoAndStop(2);
        } else {
            _root.menu2[i].MarkArmDef.gotoAndStop(1);
        }
    }
    resize();
}
function resize () {   // resize という関数を定義
    for (i=0; i<_root.maxMenu; i++) {
        eval(this+"."+i)._visible = false;
    }
    for (i=0; i<DispNum; i++) {
        eval(this+"."+i)._visible = true;
    }
    MenuBottom._y = eval(this+"."+DispNum)._y;
}
```

MenuDisp では、cmdName の配列のデータ数を DispNum の中に入れ、i を 0 としておき DispNum より小さい時、MenuItemName に cmdName[i].Name を入れ、MarkArmAtk、MarkArmDef の 1 フレーム目で停止しておき、関数 resize を実行します。MarkArmAtk、MarkArmDef は武器、防具を装備したときにアイテムの横につくマークのことで、どちらも 1 フレーム目には何も表示させず、2 フレーム目でマークを表示させています。

cmdName は「メニュー」の中の 2 フレーム目で出てきます。

仮に　cmdName = “state” とします。

state は「シーン 1」の 4 フレーム目で

```
state = new Array();
for (i=0; i<=2; i++) {
      state[i] = new Object();
}
state[0].Name = "つよさ";
state[0].ExecNo = 4;
state[1].Name = "どうぐせいり";
state[1].ExecNo = 5;
state[2].Name = "やめる";
state[2].ExecNo = 10;
```

となり、配列の場合での length は配列のデータ数になるので、cmdName.length は 3 となります。(state[0]、state[1]、state[2])
MenuItemName に「つよさ」「どうぐせいり」「やめる」を入れ装備した時のマークを非表示にし、resize を実行します。関数 resize のルーチンについては後で説明します。

MenuDisp2 は自分の持っているアイテムを見る場合に使います。
Arm1、Arm2 は、武器、防具を装備していることを意味し、

```
        if (_root.item[i].Arm1) {
            _root.menu2[i].MarkArmAtk.gotoAndStop(2);
        } else {
            _root.menu2[i].MarkArmAtk.gotoAndStop(1);
        }
```

は自分の持っているアイテムの中で装備している武器があれば、その横に装備マークをつけ、そうでなければマークをつけないことを意味しています。防具についても同様で実行し、その後 resize を行ないます。

resize ではメニューを整形しています。
maxMenu より小さい時、ムービークリップ「メニューアイテム」のインスタンス名「0」「1」「2」-- 「24」をまず非表示にします。 maxMenu は「シーン 1」の 4 フレーム目で「25」と定義しています。
次に i が DispNum より小さい場合、インスタンス名 i 番目までを表示します。
state の場合だと、DispNum は 3 になるので「メニューアイテム」のインスタンス名「0」「1」「2」を表示して「3」から「24」までは非表示になります。
しかしこのままだと「2」までが表示されて下に空きができてしまいます。
そこでムービークリップ「menubottom」をつくりインスタンス名を「MenuBottom」とします。そして MenuBottom の y 座標を DispNum の下にあわせます。

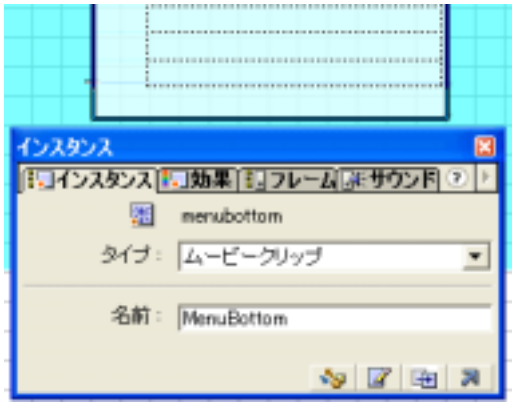


図 5―3―5　ムービークリップ「menubottom」

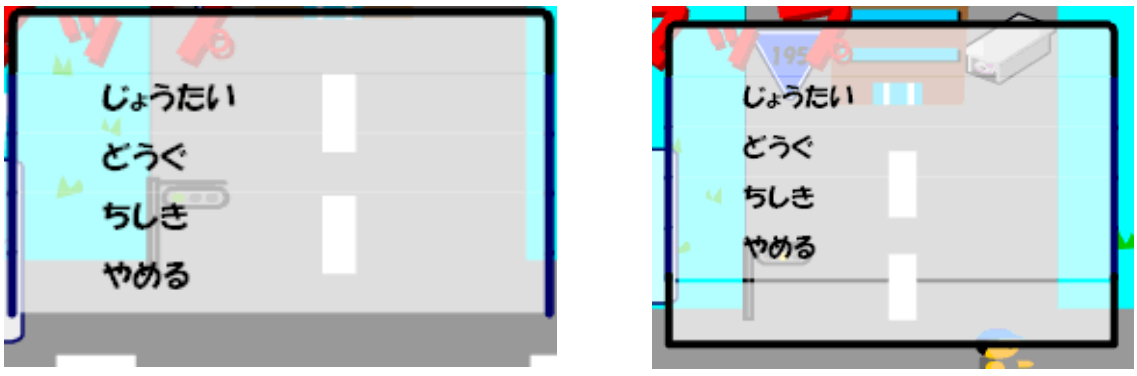


図 5―3―6　ゲーム中での「メニュー」の表示
(左が「menubottom」がない状態、右が「menubottom」がある状態)

「メニュー」の 2 フレーム目では ”ExecNo”の処理について記しています。
次に示すアクションスクリプトは 2 フレーム目に記述されているスクリプトの一部です。

```
ExecNo = eval("_root."+cmdName)[this.MenuValue].ExecNo;
if (ExecNo eq 1) {   //  「どうぐ」を選んだとき
        for (i=0; i<_root.maxMenu; i++) {
                eval("_root.menu2."+i).gotoAndStop(1);
        }
        _root.GameItemMenu();
        _root.menu2.cmdName = "itemMenu";
        _root.menu2.MenuDisp2();
        _root.menu2._visible = true;
        _root.NowMenu = "menu2";
} else if (ExecNo eq 2) {   //  「じょうたい」を選んだとき
        for (i=0; i<_root.maxMenu; i++) {
                eval("_root.menu2."+i).gotoAndStop(1);
        }
        _root.menu2.cmdName = "state";
        _root.menu2.MenuDisp();
        _root.menu2._visible = true;
        _root.NowMenu = "menu2";
}
```

まずはじめにこのフレーム内で使用される ExecNo を定義します。

```
eval("_root."+cmdName)[this.MenuValue].ExecNo;
```

cmdName はそれぞれの場所で定義され、MenuValue は「メニューアイテム」のインスタンス名、ExecNo は 3 や 4 フレームで生成された配列の ExecNo を意味します。そしてそれらの値を ExecNo の中に入れています。

次に ExecNo が 1 の場合の処理について説明します。
「menu2」に 25 個の「メニューアイテム」を出します。「menu2」は「シーン 1」に配置したムービークリップ「メニュー」のインスタンス名です。「メニュー」のインスタンス名は他にも、「menu1」「menu3」「menu4」があります。

GameItemMenu は「シーン 1」の 3 フレーム目に関数で定義されており、次のようになります。

```
function GameItemMenu () {
    itemMenu = new Array();
    for (i=0; i<item.length; i++) {
        itemMenu[i] = new Object();
        itemMenu[i].No = item[i].No;
        itemMenu[i].Name = item[i].Name;
        itemMenu[i].ExecNo = item[i].ExecNo;
    }
    itemMenu[i] = new Object();
    itemMenu[i].Name = "やめる";
    itemMenu[i].ExecNo = 10;
}
```

GameItemMenu では、自分の持っているアイテムの情報を新たに登録します。そして cmdName を ”itemMenu”として MenuDisp2 の処理を行い、「menu2」を表示させ、NowMenu を “menu2”とします。NowMenu はシーン 1 の 4 フレーム目で定義されており、初めて登場した時は ”menu0” となっています。

ExecNo eq 1 は、「どうぐ」コマンドを選んだときの処理を意味し、「どうぐ」を選ぶと「menu2」に現在自分が持っている道具を表示します。もしここで武器や防具などを装備しているとその横に装備マークがつきます。

ExecNo が 2 の時は、cmdName を ”state” にし、MenuDisp の処理を行い、「menu2」を表示させ、NowMenu を “menu2” にします。

ExecNo eq 2 は、「じょうたい」コマンドを選んだときの処理を意味し、「じょうたい」を選ぶと、「menu2」に「つよさ」「どうぐせいり」「やめる」コマンドを表示します。

「メニュー」の 2 フレーム目のフレームアクションは付録で紹介しています。

第 6 章　総括

今回自分でゲームを作ってみて、ゲームの制作がいかに大変な作業であるかということ、同時に始め自分が描いていた理想を現実のものにすることがどれだけ難しいかということを学びました。

特に「メニュー」部分のプログラム作成では、１つコマンドを追加するだけでもうまくいかなければ追加した部分だけが機能しなくなるだけではなくゲーム全体がおかしくなったり、記述したスクリプトの量が多くなってくるとどこで間違っているのかを見つけるのにも苦労しました。

またホームページを作成し Web にアップする作業では、自分のパソコンではうまくリンクが張られていても、いざ Web にアップしてみたら他のサイトへのリンクだけではなく、自分のサイト内のリンクでさえうまくいかなかということもありました。

プログラムを作成し、間違いを見つける作業は大変だったが、それでもゲームが完成した時は本当によかったと思えました。今回の研究で、ゲームを作ることの難しさと同時に出来上がった時の達成感を体験できたことが自分にとってはよい経験になりました。そして今回学んだことをこれからも忘れずにすごして生きたい。

参考文献

1. リンダ･ワインマン&ガロ･クリーン　共著　「Dreamweaver 3 ハンズ･オン･トレーニング」
2. 外間かおり　著　「DREAMWEAVER3 スーパーリファレンス」
3. 杉山　敦　著　「ウェブデザイナーのための DREAMWEAVER3 & FIREWORKS3」
4. future brain project　著　「DREAMWEAVER & FIREWORKS　プロフェッショナル・ウェブ・デザイン」
5. Robin Willians & John Tollett 著　「ノンデザイナーズウェブブック 2001」
6. Web&HP 研究会　編著　「Fireworks3Web 画像作成スーパーテクニック」
7. 外間かおり+なかひらまい　著　「Flash5 スーパーリファレンス」
8. 保坂庸介　著　「Flash MX 400 シンボル･テンプレート･ブック」
9. 境　祐司　著　「速習 Web デザイン Flash5」
10. C&R 研究所　著　「アッと驚く達人の技　Flash5　上級テクニック集」
11. マーク･クラークソン　著　「Flash アニメーターズガイド~カートゥーンキャラクタを動かそう!」
12. A.e.Suck　著　「Flash アニメーション完全攻略」
13. 大重美幸　著　「Flash ActionScript サンプル集」
14. 柴田忠浩+広石里香　著　「おしえて!!　Macromedia Flash5 アクションスクリプト 」
15. 森　巧尚　著　「GO! GO! FLASH5　ゲームを作ろう編」
16. 上野　亨　著　「FLASH ActionScript バイブル」

謝辞

今回の研究と論文の作成にあたり、終始丁寧なご指導とご教示を賜りました高知工科大学工学部電子・光システム工学科綿森道夫助教授に深い感謝の意を表します。

また、高知工科大学電子･光システム工学科在学中にご指導を賜った原央学科長に心から感謝致します。

最後に高知工科大学電子･光システム工学科在学中、本研究の実験遂行、各過程で終始ご厚意、ご協力を頂きました高知工科大学電子･光システム工学科、河津哲教授・成沢忠教授・平木明夫教授・矢野政顕教授・神戸宏教授・畠中兼司教授・河東田隆教授・西本俊彦教授・山本哲也教授・野中弘二教授・橘昌良助教授、八田章光助教授・井上昌昭助教授・関口晃司助教授・笹原泰講師・武田光由実験講師・西田謙助手の方々には重ねて感謝の意を述べさせていただきます。