

iアプリ Java ゲームプログラミング

総集編 シューティングゲームの作成

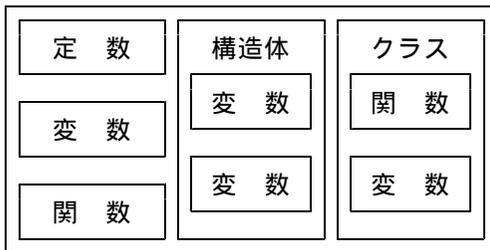
Java

Javaとは、Sun Microsystems社が開発したオブジェクト指向プログラミング言語で、その実行環境も含めてJavaと呼びます。

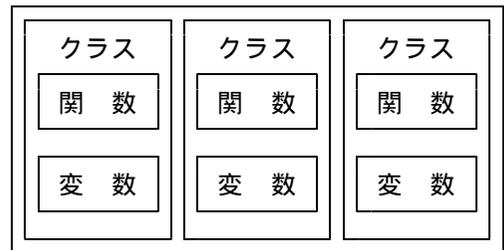
Javaは、どのような環境でも実行できることを目標としており、仮想マシンが存在すれば、OSに依存することなくコードを実行できることが特徴です。OSに依存しないコードが書けることから、サーバー用のアプリケーションや、組み込みソフトの実行環境として携帯電話に普及しています。

Javaの特徴

Javaは、C++の優れた点だけを引き継ぎ、オブジェクト指向に不要なものや危険性の高いものを削り、シンプルで安全性の高い言語を目指しています。たいていの言語が文字とファイルの入出力しか言語仕様には含んでいないのに対し、Javaでは画像、GUI、ネットワークなどの膨大なクラスライブラリを言語仕様として標準サポートしています。



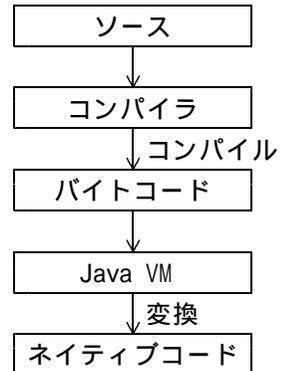
C++で作成されたプログラム
(構造化 + オブジェクト指向プログラミング)



Javaで作成されたプログラム
(完全なオブジェクト指向プログラミング)

Javaの仕組み

Javaで記述されたプログラムは、コンパイラによりバイトコードにコンパイルされます。バイトコードは、JavaVM(Java Virtual Machine)という仮想マシン用のコードです。JavaVMは、バイトコードを1つ1つ環境に合うように解釈しながら実行します(バイトコードインタプリタ方式)。直接実行できるネイティブコードに比べ実行速度は劣りますが、Windows, Macintosh, Linuxなどさまざまな環境のものが提供されており、同じJavaプログラムをOSやCPU、表示機能、ネットワークなどの違いを超えて、異なる環境で動かすことができるようになっています。



iアプリ

iアプリは、NTTドコモのiモード端末機でダウンロードして利用できるアプリケーションです。ゲームや地図情報、株価情報などのコンテンツが携帯電話で楽しめます。

携帯電話は、PCと比較してハードウェアの性能上、厳しい制約があり、通常のJavaより機能は削られています。

iアプリ開発環境

iアプリを作成するには、最低限、以下のソフトウェアが必要になります。これは無償で入手することができます。

- Java2 SDK Standard Edition (1.4.2以降)
- i ppli Development Kit for Star

JDK

JDKは、Java言語でプログラミングを行なう際に必要な最低限のソフトウェアのセットです。Javaの開発元であるSun Microsystems社が開発、配布しています。コンパイラ、デバッガ、クラスライブラリ、JavaVMなどが含まれています。

JDKはOSごとに専用のものが存在し、どれも無償でダウンロードして入手することができます。なかでも、Windows, Linux, SolarisのJDKは、ORACLE社(<http://www.oracle.com/technetwork/java/index.html>)が提供しています。Macintoshや商用UNIXなども、OSのメーカーが提供しています。

i Appli Development Kit for Star

Starとは、FOMA2008年秋冬モデル(x-0xA)から採用された、これまでのDoJaにかわる、NTTドコモが提供するiアプリ用のJava拡張ライブラリです。Java2ME CLDC 1.1を拡張し、スクラッチパッドや携帯電話のボタンの処理機能などが定義されています。

Starは、iアプリ対応携帯電話では共通して使用できますが、auのBREWやソフトバンクのsアプリでは使用できません。基本的に上位互換ですが、たとえば、Star-1.1は、Star-1.0を拡張したのですが、Star-1.1で追加されたライブラリはStar-1.0端末では使用することができません(エラーになり、強制終了します)。

Starは、NTTドコモのホームページ(<http://www.nttdocomo.co.jp/service/imode/make/content/iappli/tool/star/index.html>)からダウンロードすることができます。

Eclipse

Eclipseは、統合ソフトウェア開発環境(IDE)の一つで、Java開発者を中心に急速に普及しています。ソフトウェア開発ツールの共通プラットフォームの標準になるといわれています。ソースコードが公開されており、無償で入手・改変・再配布できるようになっています。大手ソフトウェアベンダの中には、自社の開発ツール製品にEclipseを組み込み、Eclipseに追加する形で自社独自部分を提供するという形の製品を発売するところも現れています。

携帯電話の仕様

携帯電話の各機種の様子は、以下のようになっています。

機種	プロファイル	JAR容量	スクラッチパッド	Canvasクラス解像度	浮動小数点对応
901i	DoJa4.0	100KB	400KB	240 × 240	
903i	DoJa5.0	合わせて1024KB		240 × 240	
905i	DoJa5.1	合わせて1024KB		240 × 240	
x-0xA	Star-1.0	合わせて2048KB		480 × 480	
x-0X	Star-1.1	合わせて2048KB		480 × 480	
x-0xB	Star-1.2	合わせて2048KB		480 × 480	
x-0xB	Star-1.3	合わせて2048KB		480 × 480	

Canvasクラスの解像度は、ADFのDrawAreaキーに何も指定しなかった場合

容量も解像度も、901iシリーズ以降は以前よりも大幅に強化されていますので、20年前のスーパーファミコン以上のゲームも作成できます。PS3やPCとは比べものにならないかもしれませんが、3Dゲームを作成することも出来ます。しかし、iアプリでは、JavaVMを稼働し、バイトコードを解釈しながら実行するので、処理が重くなることもあります。

変数

Javaの変数には、次の8つの型があります。

型名	説明	値の範囲
boolean	真偽値を格納	true(成り立つ), false(成り立たない)
char	unicode文字を格納	¥u0000 ~ ¥uffff
byte	1バイト整数	-128 ~ 127
short	2バイト整数	-32,768 ~ 32,767
int	4バイト整数	-2,147,483,648 ~ 2,147,483,647
long	8バイト整数	-92,233,372,036,854,775,808 ~ 92,233,372,036,854,775,807
float	単精度浮動小数点(4バイト)	±10の±38乗(有効桁数9桁)
double	倍精度浮動小数点(8バイト)	±10の±308乗(有効桁数16桁)

Javaで使用するおもな演算子は、以下のものです。

演算子名	記号	名 前	記述例	機 能
算術演算子	++	インクリメント	a++	値に1を加える
	--	デクリメント	a--	値から1を引く
	+	正符号	+a	正符号
	-	負符号	-a	負符号
	*	乗算	a * b	乗算
	/	除算	a / b	除算
	%	剰余	a % b	割り算の余りを求める。
	+	加算、文字列の連結	a + b	加算、両辺が文字列なら連結する
-	減算	a - b	減算	
演算子名	記号	名 前	記述例	機 能
ビット演算子	<<	左シフト	a << b	ビットを左にシフトする
	>>	算術右シフト	a >> b	ビットを右にシフトする
	>>>	論理右シフト	a >>> b	ビットを右にシフトする。先頭は0で埋められる
演算子名	記号	名 前	記述例	機 能
関係演算子	==	等値	a == b	両辺が等しければtrue、そうでなければfalseとなる
	!=	非等値	a != b	両辺が等しくなければtrue、そうでなければfalseとなる
	<	左不等	a < b	左辺より右辺が大きければtrue、そうでなければfalseとなる
	<=	等価左不等	a <= b	左辺が右辺以上ならtrue、そうでなければfalseとなる
	>	右不等	a > b	左辺より右辺が小さければtrue、そうでなければfalseとなる
	>=	等価右不等	a >= b	左辺が右辺以下ならtrue、そうでなければfalseとなる
演算子名	記号	名 前	記述例	機 能
論理演算子	&&	条件 AND	a && b	両辺がtrueのときのみtrueとなる
		条件 OR	a b	少なくとも一方がtrueならtrueとなる
演算子名	記号	名 前	記述例	機 能
代入演算子	=	単純代入	a = b	右辺の式の値を左辺に代入する
	/=	除算代入	a /= b	a = a / bと同じ
	%=	剰余代入	a %= b	a = a % bと同じ
	*=	乗算代入	a *= b	a = a * bと同じ
	+=	加算代入	a += b	a = a + bと同じ
	-=	減算代入	a -= b	a = a - bと同じ

制御文

制御文とは、プログラムの流れを制御するための文です。Javaでは上から順番にプログラムが実行されますが、制御文を使うことにより、プログラムを分岐させたり、反復したりすることができます。

分岐させる制御文としてif-else、switch-caseがあり、反復(ループ)させる制御文としてwhile、do-while、forがあります。

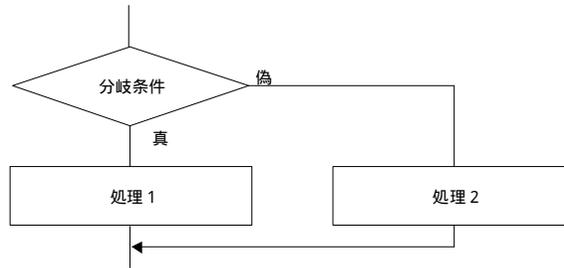
1 . if-else

if-else文は、条件が成り立つ場合は「処理1」が行われ、成り立たない場合は「処理2」が実行されます。

```

if(条件式) {
    条件式を満たしたときに実行する処理
} else {
    条件式を満たさなかったときに実行する処理
}

```



2 . switch-case

switch文は、値によって実行する処理を多分岐させることができます。

```

switch(変数) {
case 定数式 1 :
    処理 1
    break;

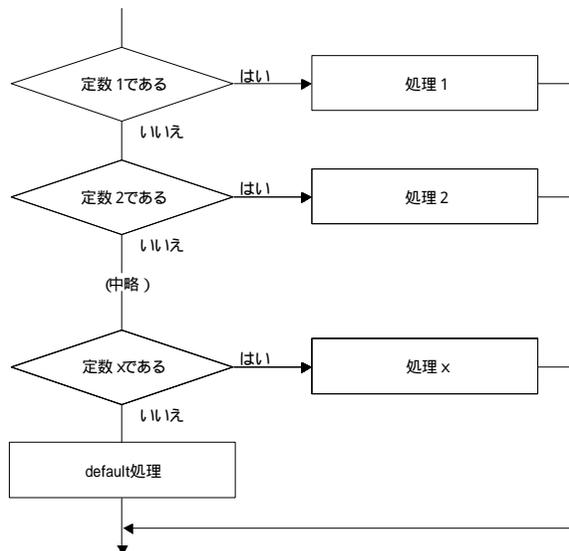
case 定数式 2 :
    処理 2
    break;

(中略)

case 定数式 x :
    処理 x
    break;

default :
    すべての定数式を満たさなかったときの処理
    break;
}

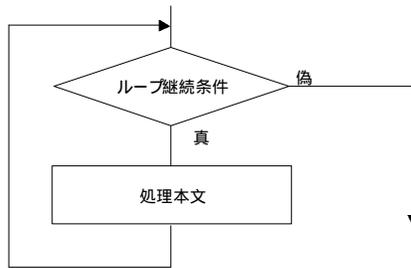
```



3 . while

while文は、ループに入る前に条件判定を行い、条件が成り立つ間処理を反復します。

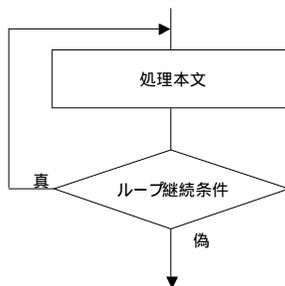
```
while(ループ継続条件) {
    処理本文
}
```



4 . do-while

do-while文は、ループの最後に条件判定を行い、条件が成り立つ間処理を反復します。

```
do {
    処理本文
} while(ループ継続条件);
```

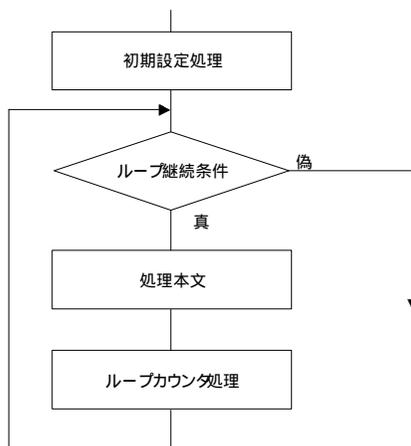


5 . for

for文は、条件が成り立つ間、処理を反復させることができます。

```
for(初期設定処理; ループ継続条件; ループカウンタ処理) {
    反復させたい処理
}
```

for文は、指定した回数だけ反復させたいときに使用します。反復回数を管理するために、ループカウンタという変数を使用します。



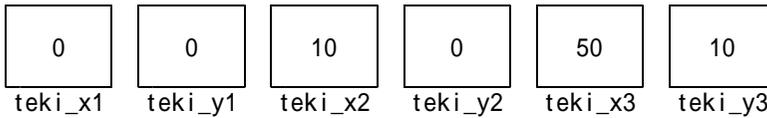
配列

データは変数に入れて扱いますが、1つのデータにつき1つの変数を宣言する必要があります。しかし、この方法ではプログラム作成上、困難になる場合があります。

たとえば、複数の敵がいて、それぞれの座標が、teki_x1, teki_y1, teki_x2, teki_y2...という変数に格納されているとしましょう。

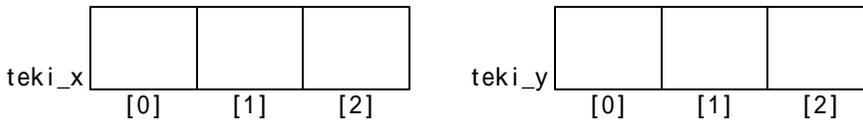
```
int teki_x1 = 0, teki_y1 = 0; // 敵No.1
int teki_x2 = 10, teki_y2 = 0; // 敵No.2
int teki_x3 = 50, teki_y3 = 10; // 敵No.3
```

これは、以下のように個々の変数を用意していることになります。



それでは、敵の数が多くなったらどうでしょう。たとえば、敵の数が1000になったとしたら、teki_x 1000とteki_y 1000まで2000個もの変数を定義しなければならなくなり、それはかなり面倒です。

ここで、配列の出番です。配列は、「個々の変数をひとかたまりにしたもの」です。配列を使うと、同じ型のデータをまとめて保持できるようになり、プログラムの煩雑さから解放されます。配列を図で表現すると、以下のように変数が連続しているイメージになります。上の図との違いは、変数が連続している点にあります。



配列とは、まとめて箱を確保し、位置(先頭からの距離)を指定してアクセスできる変数です。

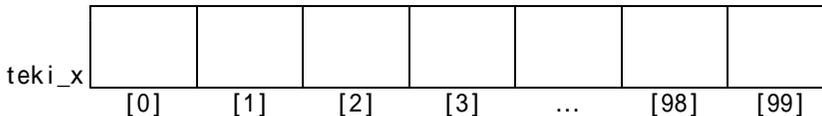
配列の定義

変数を使用するには定義が必要であったのと同じように、配列を使うためにも定義が必要です。

```
int[] teki_x = new int[100]; // int teki_x[] = new int[100]でも可
```

これは、teki_xという名前のint型の配列を定義しています。変数名の後に、new演算子と型名につづき、角括弧[]でくくって必要な個数を記述することで、配列の数(これを要素数といいます)を定義することができます。この例ではint型の100個のデータが格納できる配列を定義しています。

配列を定義すると、メモリには以下の図のような変数の格納領域ができあがります。ちょうど、teki_x[0], teki_x[1], ..., teki_x[99]という名前の100個の変数が並んだ形になります。先頭がteki_x[1]ではなく、teki_x[0]になっていることに注意してください。



配列の添字

配列の個々の変数(これを配列の要素と呼びます)は、括弧[]を使ってアクセスすることができます。たとえば、上記の配列teki_xでは、teki_x[0]やteki_x[1]とすることで各要素にアクセスできます。この括弧でくくられた数字の部分を配列の添字と呼びます。Javaで配列の要素を扱う場合、次のことを注意してください。

配列の最初の要素は添字が0であり、最後の添字は「配列の要素数 - 1」である

配列の最初の要素には、添字を0にして(たとえば、teki_x[0])としてアクセスします。その次が1でアクセスします。0から始まることに注意してください。

100個の要素数の場合、最後の100番目の要素にアクセスするためには、添字を99とします。100にしてはいけません。0から始まるので、そこから100数えると最後は99になるわけです。また、添字には変数を使用することもできます。たとえば、

```
int[] array = new int[100];
int i = 9;
int j = array[i]; // array[9]と同じ意味
```

というように変数を使ってアクセスすることもできます。

プロジェクトの作成

Eclipseでは、プロジェクトという単位でプログラムを管理します。i アプリでは、ひとつのi アプリに対してひとつのプロジェクトを作成します。

プロジェクトは、Javaソースファイルやリソースと呼ばれる各種画像や音楽ファイルなどをまとめて管理するものです。プロジェクトの実体は、プロジェクトフォルダ以下に作成される各種フォルダです。フォルダの構成は、以下のようになっています。

フォルダ	用途
bin	実行ファイル(.jar)が生成されるフォルダ
classes	コンパイルしたファイルが保存されるフォルダ
res	画像や音楽などのリソースを置くフォルダ
sp	データを保存するときに用いるスクラッチパッドの内容が保存されるフォルダ
src	ソースファイルを保存するフォルダ

課題

Eclipseを起動し、i アプリ用のプロジェクトを作成しましょう。

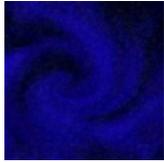
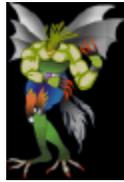
デスクトップにあるEclipseのアイコンをダブルクリックして起動し、メニューから「ファイル」「新規」「プロジェクト」を選択します。

「新規プロジェクト」の画面が表示されます。i アプリを作成する場合は、左側のツリーで「Java」を選んで、右側のプロジェクト一覧からは「DoJa-4.0プロジェクト」を選びます。そして「次へ」をクリックします。

プロジェクト名を入力する画面が表示されるので、プロジェクトの名前「ShootingGame」を入力して「終了」をクリックします。ここで入力した名前のフォルダが作成されます。

画像の準備

今回作成するシューティングゲームでは、以下のような画像を使用します。

目的画像	背景	プレイヤー	敵	プレイヤー弾	敵の弾	おしまい
						
ファイル名	BG1.jpg	PLY00.gif ~ PLY03.gif	ENM00.gif ENM01.gif	SHT00.gif ~ SHT03.gif	ENMSHT.gif	OSHI.gif
サイズ	480 x 480	42 x 55	113 x 168	24 x 22	18 x 32	157 x 37
説明	背景となる画像です。	プレイヤーが操作するキャラクタの画像です。アニメーション用に、4コマ分用意してあります。	敵の画像がENM00.gif、ダメージを与えたときの画像がENM01.gifです。	プレイヤーが発射する魔法弾です。アニメーション用に、4コマ分用意してあります。	敵が発射する弾です。	ゲームが終了したときに表示する画像です。

課題

シューティングゲームで使用する画像を作成し、プロジェクトのリソースを置くためのフォルダにコピーしましょう。

クラス

Javaでは、「クラス」という部品を設計し、クラスを組み合わせることでプログラムを構築します。i アプリ対応ゲームの場合は、StarアプリケーションのベースとなるStarApplicationクラスと、画面に描画するためのPanelクラスまたはCanvasクラスのどちらかをもとに(継承して)クラスを作成します。通常、ゲームプログラムでは、Starアプリケーションクラスとキャンバスクラスを作成します。

クラスとは、オブジェクト指向プログラミングにおいて、データ(変数と定数)とその操作手順であるメソッド(関数)をまとめたものをいいます。オブジェクト指向プログラミングでは、クラスを定義することが基盤となっています。また、あるクラスを他のクラスに受け継がせることを「継承」といい、すでにあるクラスを再利用することにより、コードを大幅に削減することができます。

StarApplicationクラスとは、i アプリ(Starアプリ)の雛形を定義したクラスです。i アプリを作成するには、StarApplicationクラスが持っているさまざまな能力を継承した新しいクラスを作成します。StarApplicationクラスは、そのままでは使用できないようになっているので、このクラスを継承して新しいクラスを作成しなければなりません。

Canvasクラスとは、i アプリで文字や図形、画像といったグラフィックを描画するためのクラスです。描画以外にも、キーの状態を取得する機能も提供しているので、まさにゲーム制作のためのクラスといえます。Canvasクラスも、そのままでは使用できないようになっているので、継承した新しいクラスを作成しなければなりません。このクラスに、ゲームの処理を記述します。

上記2つのクラス以外にも、本格的にオブジェクト指向を用いれば、さまざまなクラスを作成していくことになります。

課題

i アプリでゲームを作成するのに必要な2つのクラスを作成し、i アプリを実行してみましょう。

(1) i アプリケーションクラスを作成します。

「パッケージ・エクスプローラ」の一番上のプロジェクト名「ShootingGame」を右クリックし、メニューから「新規(W)」 「クラス」を選びます。

「新規 Javaクラス」ウィンドウが表示されるので、「名前(M)」と「スーパークラス(S)」を入力します。「名前」はクラス名です。「MyStarAppli」とします。また、スーパークラス(もともとなるクラス)には「com.docomostar.StarApplication」(NTTドコモのStarApplicationクラス)と入力します。

「final(L)」と「継承された抽象メソッド」にチェックを入れ、「終了」ボタンを押します。



クラス名「MyStarAppli」を入力します

「final」にチェックを入れると、少しだけサイズを削減できます

継承するクラス名の正式名称
「com.docomostar.StarApplication」
を入力します。

「継承された抽象メソッド」にチェックを入れます。必要なメソッドが自動的に作成されます。

MyStarAppliクラスのソースファイルが作成されます。

(2)キャンバスクラスを作成します。

「パッケージ・エクスプローラー」の一番上のプロジェクト名「ShootingGame」を右クリックし、メニューから「新規(W)」 「クラス」を選びます。

「新規 Javaクラス」ウィンドウが表示されるので、「名前(M)」と「スーパークラス(S)」を入力します。名前を「MyCanvas」、スーパークラスは「com.docomostar.ui.Canvas」と入力します。「final(L)」と「継承された抽象メソッド」にチェックを入れ、「終了」ボタンを押します。

MyCanvasクラスのソースファイルが作成されます。

(3) i アプリの設定を行います。

「パッケージ・エクスプローラー」で一番上のプロジェクト名をクリックし、メニューから「プロジェクト(P)」 「Star-1.0(S)」 「ADF/トラステッド動作設定(C)」を選択します。

「プロジェクト設定」ウィンドウが表示されます。ここで入力しなければならないのは、一番上の「AppName」と中盤やや上の「AppClass」です。

「AppName」は、アプリケーションの名前「ShootingGame」を入力します。ここで指定した名前は、実際に i アプリをダウンロードした際、i アプリ一覧に表示されます。

「AppClass」は、(1)で作成したアプリケーションのクラス名を入力します。今回は「ShootingGame」と入力します。

正しく入力できたら、「OK」ボタンを押します。

(4) i アプリを実行しましょう。

メニューから「実行」 「実行」を選択します。

「実行」ウィンドウが表示されるので、左側のツリーで「Star-1.0アプリケーション」を右クリックし、メニューから「新規」を選びます。

新しい実行設定が作成されます。「名前」はそのままで構いません。「実行」ボタンをクリックすると、実行が始まります。

保存していないファイルがある場合は「リソースの保管」ウィンドウが表示されます。保存するファイルにチェックを付けて「OK」ボタンを押します。

エミュレータのウィンドウが表示され、i アプリが実行されます。

「PWR/HDL」ボタンを押すと終了します。

次回からは、アイコン○をクリックするだけで実行できます。

メソッド

メソッドとは、クラスに備えられた機能のことをいいます。逆に言えば、クラスに機能を追加したいときはメソッドを定義する、ということです。

MyStarAppliクラスには、アプリケーションを開始するためのstartedメソッド、MyCanvasクラスには、画面の描画を行うpaintメソッドがすでに作成されています(ただし、処理は記述されていません)。言い換えれば、MyStarAppliクラスには、アプリケーションを開始する機能、MyCanvasクラスには、画面の描画を行う機能があると言えます。

i アプリでゲームを作成する場合は、キャンバスクラスに機能を追加していきます。つまり、MyCanvasクラスに必要なメソッドを追加していきます。

ゲームを初期化する機能、メインループ、プレイヤーや敵キャラクターの処理、キャラクターを描画する機能などは、メソッドとしてキャンバスクラスに記述します。

キャンバスクラスに、以下の機能を追加しましょう。

- ・実行.....run メソッド
- ・初期化.....initメソッド
- ・内部処理...procメソッド
- ・描画.....drawメソッド

(1) MyCanvasクラスのソースコードを以下のように変更しましょう。

- MyCanvas.java -

```
import com.docomostar.ui.*;

public final class MyCanvas extends Canvas {

    // --- 定数の定義(ゲームに必要な定数を以下に定義します)

    // --- 変数の定義(ゲームに必要な変数を以下に定義します)

    // --- メソッドの定義(ゲームに必要なメソッドを以下に定義します)
    // --- 実行
    void run() {
    }

    // --- 初期化
    void init() {
    }

    // --- 内部処理
    void proc() {
    }

    // --- 描画
    void draw() {
    }

    public void paint(Graphics arg0) {
        // TODO 自動生成したメソッド・スタブ
    }
}
```

Eclipseでは、プログラムの入力中に補完候補が現れることがあります。一覧から選択して「Enter」キーを押すと、自動的に選んだ項目が入力され便利です。このような補完機能を活用することで、単純な入力ミスを防ぐことが可能です。補完候補は「Ctrl」+「Space」キーで手動表示させることもできます。

エラーが検出されたときには、画面左にエラーマーク表示されます。ここで、エラーマークをクリックすると、エラーの修正候補が表示されます。

このように、Eclipseは、非常に強力な入力支援機能を持っています。

各メソッドの役割は、以下のようになります。

- ・runメソッドは、ゲームを実行するメソッドです。このメソッドにゲームの中心となる処理を記述します。
- ・initメソッドは、ゲームを初期化するメソッドです。変数や画像など、ゲームを初期状態に設定するための処理を記述します。
- ・procメソッドは、ゲームの内部処理を行うメソッドです。キャラクターの座標や状態を変更するための処理を記述します。
- ・drawメソッドは、描画を行うメソッドです。内部処理で変更された状態を画面に表示するための処理を記述します。

(2) MyStarAppliクラスのソースコードを以下のように変更しましょう。

- MyStarAppli.java -

```
import com.docomostar.StarApplication;
import com.docomostar.ui.Display;

public final class MyStarAppli extends StarApplication {

    public void started(int arg0) {
        MyCanvas canvas = new MyCanvas(); // キャンバスの生成
        Display.setCurrent(canvas); // キャンバスをディスプレイに設定
        canvas.run(); // キャンバスのrunを呼び出し、ゲームを実行する
        terminate(); // runメソッドの実行が終わったら、アプリ終了
    }
}
```

startedメソッドは、i アプリが実行されたときに最初に実行されるメソッドです。startedメソッドで行うのは、キャンバスの作成、ディスプレイへの設定、ゲームの実行です。

i アプリでは、詳細な処理はキャンバスクラスに記述するのが一般的なので、アプリケーションクラスには、プログラムの起動、実行、終了のみを記述します。

(3) プログラムが入力できたら、「Ctrl」+「s」キーを押して保存しましょう。この際、自動的にコンパイルされます。

(4) アイコン○をクリックし、実行しましょう。

メインループ

ゲームプログラムは、「ものの状態の集まり」と考えることができます。キャラクターの座標やパラメータ、背景、画面エフェクトなどすべて、それぞれ「もの」の状態です。ゲームプログラムとは、このいろいろな「もの」の状態を時間に沿って更新し、それを画面に描画するプログラムと考えることができます。

つまり、ゲームプログラムの処理を非常に単純化すると、

- ・「もの」の状態を更新(内部処理)
- ・「もの」を画面に描画(描画処理)

という2つにまとめることができます。

この2つの処理の繰り返しがメインループになります。このループ1回ぶんを「1フレーム」と呼びます。1フレームごとに、キャラクタの位置や状態、背景やそのほかの表示状態などを更新し、その状態を反映して描画を行う、という処理を繰り返していきます。

フレームレート

ゲームプログラムでは、メインループ(正確には画面の更新)を一定の間隔で実行します。実行速度は環境によって異なるので、速い環境ほど高速に動作できます。

1秒間に何回画面を更新するのかをフレームレート(FPS:Frame Per Second)と呼びます。60FPSの場合、1秒間に60回画面を更新しているという意味になり、1フレームを約16.67ミリ秒の間隔で表示しています。

一般的な家庭用ゲームでは60FPSです。携帯電話では、液晶の反応速度の兼ね合いもあり、30FPS程度にしておきます。この数値は、表示する画像の大きさや数が増えるほど、または高性能な描画機能を使用すると目に見えて落ちていきます。



課題

(1)キャンバスクラスにメインループを作成します。

メインループは、キャンバスクラスのrunメソッド(ゲームを実行するためのメソッド)に記述します。メインループに入る前に、initメソッドを呼び出して変数を初期化しておきます。メインループでは、内部処理と描画処理を交互に行い、ゲームを進めていきます。

「？」を埋め、runメソッドを以下のように変更しましょう。

```

// --- 実行
void run() {
    init(); // 初期化

    // メインループ
    while(true) {
        ???(); // 内部処理
        ???(); // 描画
    }
}

```

(2)フレームレートを制御する機能を追加します。

フレームレートを保つには、メインループが一定の間隔で回るようにします。メインループの処理時間は、その都度変わるので、メインループ1回ごとに処理時間を計り、それが1フレームの間隔未満の場合は、時間に余裕があるので、その分だけプログラムを休止させます。

休止時間は、以下の式で計算するものとします。

休止時間 = 1フレームの間隔 - 1フレームを処理するのにかかった時間
 (休止時間が負の場合は、処理落ちになります)

1フレームの間隔 = 1000 / FPS (1秒 = 1000ミリ秒)

1フレームを処理するのにかかった時間 = 現フレーム終了時間 - 前回のフレーム終了時間
 (前回のフレーム終了時間 = 現フレーム開始時間)

キャンバスクラスに、フレームレートを制御するための定数を定義します。定数FPSおよびINTERVALを「public final class MyCanvas extends Canvas {」の直下に追加しましょう。

```
public final class MyCanvas extends Canvas {  
  
    // --- 定数の定義(ゲームに必要な定数を以下に定義します)  
    public final static int    FPS        = 30;           // フレームレート  
    public final static int    INTERVAL  = 1000 / FPS;    // フレームの間隔(ミリ秒)  
  
    // --- 変数の定義
```

定数FPSが1秒あたりのフレーム数、定数INTERVALが現フレームから次フレームまでの間隔です。

(3)フレームレートを制御するための変数を追加します。

(4)の方法でフレームレートを制御するには、「前回のフレーム終了時間」を変数に保存しておく必要があります。long型の変数「lastTime」を「// --- 変数の定義」の下に追加しましょう。

```
// --- 定数の定義  
public final static int    FPS        = 30;           // フレームレート  
public final static int    INTERVAL  = 1000 / FPS;    // フレームの間隔  
  
// --- 変数の定義  
long    lastTime = 0;                                // 前回のフレーム終了時間  
  
// --- 実行  
void run() {
```

(5)休止時間を求め、プログラムを一時休止させるsleepメソッドを作成します。以下のプログラムをキャンバスクラスに追加しましょう。

```
// --- 描画  
void draw() {  
}  
  
// --- 休止  
void sleep() {  
    long    wait = INTERVAL - (System.currentTimeMillis() - lastTime); // 1  
    if(wait > 0) {  
        try {  
            Thread.sleep(wait); // 2  
        } catch(Exception e) {  
        }  
    }  
    lastTime = System.currentTimeMillis(); // 3  
}
```

1は、(3)の方法で休止時間を求める部分です。休止時間がある場合、2のThread.sleepメソッドでプログラムを休止させます。3は、実質的にメインループの最後に行われる処理で、フレームの終了時間を保存する処理です。System.currentTimeMillisメソッドを呼び出し、現在の時間を取得しています。

(6)メインループの最後で休止するようにします。メインループを以下のように変更しましょう。

```
// --- 実行  
void run() {  
    init(); // 初期化  
  
    // メインループ  
    while(true) {  
        ???(); // 内部処理  
        ???(); // 描画  
        sleep(); // 休止  
    }  
}
```

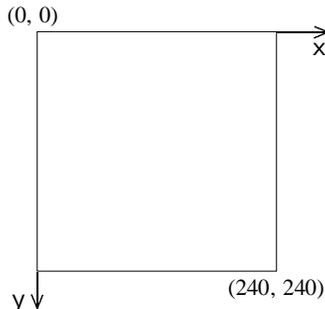
Graphicsクラス

i アプリでグラフィックを描画する場合、Graphicsクラスを用います。このクラスは、文字を描画するdrawStringメソッド、線を引くdrawLineメソッド、矩形を描くdrawRectメソッド、画像を描画するdrawImageメソッドなど、描画に関するさまざまな機能を提供しています。

また、ゲーム向けの機能もあります。2D-RPGのフィールドのような、さまざまな部品を敷き詰めて描画するdrawImageMapメソッド、画像の任意の部分を拡大縮小して描画するdrawScaledImageメソッドや、一昔前のゲーム機に搭載されていたスプライトを実現するためのSpriteクラスが導入されています。

スクリーン座標

2Dでグラフィックを描画する場合、座標の指定はスクリーン座標で行います。スクリーン座標とは、画面のピクセルに1対1に対応する座標系のことです。スクリーン座標の原点は、左上にあります。また、y軸の向きが逆になっていることも大きな特徴です。



画像の読み込み

画像を読み込むには、以下のクラスライブラリを使用します。

- MediaManager
- MediaImage
- Image

画像は、MediaManagerをとおして取得します。MediaManager.getImageメソッドに画像が保存されている場所とファイル名を指定すると、形式のチェック、アクセス可能な場所かどうかのチェックを行います。誤りがなければ、画像を管理するためのMediaImage型の変数が返されます。

扱うことのできる形式は、GIF、透過GIF、JPEGです。指定できる場所は、リソース、URL、スクラッチパッド、byte配列ですが、大文字と小文字が区別されるので、注意が必要です。

MediaImageは、画像を管理します。MediaImageが指定された場所から画像を読み込み、各機種に合わせたデータに変換します。そのほか、画像の幅、高さ、イメージデータの取得、読み込まれた画像をメモリから解放する機能を提供します。

画像を描画するには、Image型の変数をGraphicsのdrawImageメソッドに渡します。Image型の変数は、MediaImageのgetImageメソッドで取得できます。

- 画像読み込みのサンプルプログラム -

```
// 画像を読み込むのと描画するのに必要な変数の宣言
MediaImage mi;
Image img;

// 場所の指定(リソース)
mi = MediaManager.getImage("resource:///ファイル名");

// 画像の使用を宣言し、メモリに読み込む
try {
    mi.use();
} catch(Exception e) {
    // 例外処理(読み込めなかったときの処理をここに記述します)
}
```

```
// イメージの取得
img = mi.getImage();
```

useメソッドは、try-catch文で囲まないとエラーになります。try-catch文は、重大なエラーが発生したとき、どのようにするかを記述するためのものです。

画像を描画するには、GraphicsのdrawImageメソッドを呼び出します。このメソッドには、Image型の変数と描画座標(イメージの左上が基準)を渡します。

```
// 画像の描画(画面の(120, 80)に描画)
grp.drawImage(img, 120, 80);
```

画像の解放

携帯電話は容量がかなり少ないので、たくさんの画像をメモリに読み込むことはできません。不要になった画像を解放し、新しい画像を読み込むなどの工夫が必要になります。

画像の解放は、以下のような処理になります。

```
// 画像使用の終了を宣言(全機種共通)
try {
    img.dispose();

    mi.unuse();
    mi.dispose();
} catch(Exception e) {
}
img = null;
mi = null;

// ゴミ掃除(ガベージコレクション)
System.gc();
```

Image型の変数を取得したので、これをdisposeメソッドにより解放します。次に、MediaImageのunuseメソッドを呼び出してメモリから画像を解放します。最後に、disposeメソッドでMediaImageを解放します。最後のdisposeメソッドを呼ばない限り、メモリから解放されない機種もあります(プログラムを終了した場合は解放されます)。

課題

(1)描画に必要なGraphicsクラスを取得します。Graphics型の変数「grp」を「// --- 変数の定義」の下に追加し、さらにグラフィックを取得するプログラムを追加しましょう。

```
// --- 定数の定義
public final static int FPS = 30;
public final static int INTERVAL = 1000 / FPS;

// --- 変数の定義
Graphics grp = getGraphics(); // キャンバス(=ディスプレイ)のグラフィックを取得
long lastTime = 0; // 前回のフレーム終了時間

// --- 実行
void run() {
```

getGraphicsメソッドを呼び出すと、キャンバスに描画するためのグラフィック(正確にはグラフィックオブジェクト)が取得できます。このグラフィックに備えられているdrawStringメソッドやdrawImageメソッドを呼び出すことにより、キャンバスにグラフィックが描画されます。キャンバスは、(Display.setCurrentにより)ディスプレイに設定されていますので、画面に描画されるというわけです。

(2) drawメソッドを次のように変更しましょう。

```
// --- 描画
void draw() {
    grp.lock();           // 描画開始

    // 以下に、描画処理を記述します

    grp.unlock(true);    // 描画終了
}
```

i アプリでは、グラフィックのlockメソッドとunlockメソッドに囲まれた部分に描画処理を記述します。これ以外の部分では、正しく描画できなかつたり、ちらつきが起こる場合があります。

(3) 背景画像を保存するための変数を定義します。キャンバスクラスに以下の変数を追加しましょう。

```
// 背景
MediaImage  bgMi;
Image       bgImg;
```

(4) 背景画像を読み込みます。initメソッドを以下のように変更しましょう。

```
//--- 初期化
void init() {
    // 画像読み込み
    try {
        // 背景
        bgMi = MediaManager.getImage("resource:///BG.jpg"); // 読み込む画像の指定
        bgMi.use(); // 画像の読み込み
        bgImg = bgMi.getImage(); // イメージを取得
    } catch(Exception e) {
    }
}
```

「resource:///BG.jpg」が読み込む画像の指定です。大文字・小文字が区別されるので、正確に入力しましょう。「resource:///」は、場所がリソースという意味です。ここを<http://>ではじまるURLを入力することにより、インターネットから画像をダウンロードすることもできます。

(5) drawメソッドで背景を描画します。drawメソッドを以下のように変更しましょう。

```
//--- 描画
void draw() {
    grp.lock();           // 描画開始

    grp.drawImage(bgImg, 0, 0); // 背景描画

    grp.unlock(true);    // 描画終了
}
```

(6) プログラムを実行し、正しく描画されるかを確認しましょう。

(7) プレイヤーの画像を保存するための変数を定義します。キャンバスクラスに以下の変数を追加しましょう。

```
// プレイヤー
MediaImage  plyMi;
Image       plyImg;
```


キー	キーの値	値	備考
数字 0	KEY_0	0x00	b0
数字 1	KEY_1	0x01	b1
数字 2	KEY_2	0x02	b2
数字 3	KEY_3	0x03	b3
数字 4	KEY_4	0x04	b4
数字 5	KEY_5	0x05	b5
数字 6	KEY_6	0x06	b6
数字 7	KEY_7	0x07	b7
数字 8	KEY_8	0x08	b8
数字 9	KEY_9	0x09	b9
*	KEY_ASTERISK	0x0a	b10
#	KEY_POUND	0x0b	b11
	KEY_LEFT	0x10	b16
	KEY_UP	0x11	b17
	KEY_RIGHT	0x12	b18
	KEY_DOWN	0x13	b19
選択/決定	KEY_SELECT	0x14	b20
クリア	KEY_CLEAR	0x20	getKeypadState(1)

よく使うキーとその値

課 題

キーパッドの状態によって、プレイヤーを移動させましょう。

(1) プレイヤーの x 座標と y 座標を変数で管理します。プレイヤーの x 座標と y 座標を格納する変数を以下のように宣言しましょう。

```
// プレイヤー
MediaImage plyMi;
Image plyImg;
int plyX, plyY;
```

(2) initメソッドで変数を初期化し、プレイヤーの初期座標を設定します。initメソッドを以下のように変更しましょう。

```
//--- 初期化
void init() {
    // 変数初期化
    plyX = 0;
    plyY = 0;

    // 画像読み込み
    ...
}
```

x 座標(plyX)、y 座標(plyY)ともに 0 を設定しているので、プレイヤーは左上に描画されます。この値には、好きな数値を設定しましょう。

(3) drawメソッドを以下のように変更し、変数の座標にキャラクターが描画されるようにしましょう。

```
//--- 描画
void draw() {
    grp.lock(); // 描画開始

    grp.drawImage(bgImg, 0, 0); // 背景描画
    grp.drawImage(plyImg, plyX, plyY); // プレイヤー描画

    grp.unlock(true); // 描画終了
}
```

(4)この状態で実行すると、変数plyX, plyYに設定した座標にプレイヤーが描画されます。

(5)プレイヤーを処理するメソッドを作成します。以下のプログラムを適切な場所に追加しましょう。

```
//--- プレイヤー処理
void player() {
}
```

(6)内部処理procメソッドを以下のように変更し、playerメソッドが実行されるようにしましょう。

```
//--- 内部処理
void proc() {
    player(); // プレイヤー処理
}
```

(7)キーパッドの状態を調べ、キャラクターを移動させます。移動は、変数plyX, plyYを増減させることで行います。playerメソッドを以下のように変更しましょう。なお、(ア)~(エ)は、各自考えてください。

```
//--- プレイヤー処理
void player() {
    // キー入力取得
    int key = getKeyPadState();

    // 移動
    if((1 << Display.KEY_LEFT & key) != 0) // 左
        (ア)
    if((1 << Display.KEY_RIGHT & key) != 0) // 右
        (イ)
    if((1 << Display.KEY_UP & key) != 0) // 上
        (ウ)
    if((1 << Display.KEY_DOWN & key) != 0) // 下
        (エ)
}
```

(8)実行して動作を確認しましょう。

(9)キャラクターの移動範囲を画面内だけにしましょう。

・ヒント1

座標は、画像の左上を指しています。

・ヒント2

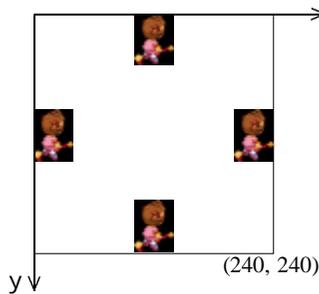
x座標の最小値は、原点の0です。この値未満になるということは、原点より左側にいるということになり、画面外にいることになります。この場合は、x座標を0にし、画面左に戻します。

・ヒント3

x座標の最大値は、画面右端の480になりそうですが、実は違います。x座標は、キャラクターの左端の座標なので、ここが480の場合は、すでに画面外に出ています。キャラクターの右端(左端 + 幅)が480を越えているかどうかを調べるか、キャラクターの左端が、480から幅を引いた値(480 - 幅)を越えているかを調べます。越えている場合は、x座標を480から幅を引いた値にします。

・ヒント4

y座標も同様に考えます。



敵の移動

敵は、プレイヤーと異なり、プログラムによって動かさなくてはなりません。動かし方はいろいろあり、一定のルートを通るだけのもの、一定のパターンで動くもの、プレイヤーや自分の状態によって異なる動作をするもの、時間によってパターンを変えるものなどさまざまな動作が考えられます。どのような動作をさせるかによって、ゲームの難易度や楽しさが変わってきます。

課題

敵の画像を読み込んで描画し、さらにプログラムによって動かしましょう。

(1) 敵の画像と座標を保存する変数を宣言します。以下のプログラムを「// プレイヤー」の下に追加しましょう。

```
// プレイヤー
Medialmage  plyMi;
Image        plyImg;
int          plyX, plyY;

// 敵
Medialmage  enmMi;
Image        enmImg;
int          enmX, enmY;

//--- 初期化
void init() {
    :
}
```

(2) initメソッドで変数を初期化し、敵の初期座標を設定します。initメソッドを以下のように変更しましょう。

```
//--- 初期化
void init() {
    // 変数初期化
    plyX = 0;
    plyY = 0;

    enmX = 127;
    enmY = 0;

    // 画像読み込み
    :
}
```

x座標(enmX)に127、y座標(enmY)に0を設定しているので、敵は右上に描画されます。この値には、好きな数値を設定しましょう。

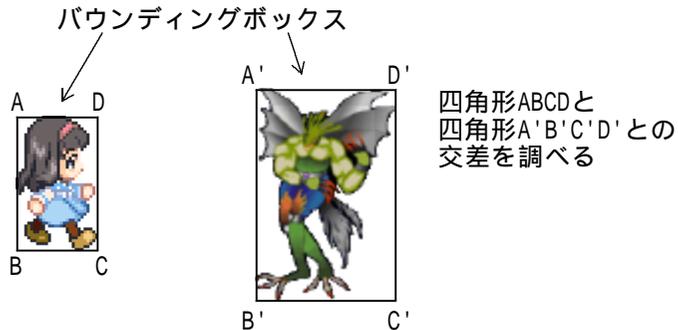
衝突検出

ゲームの中で、キャラクター同士の衝突検出(Collision Detection:衝突判定、当たり判定、ヒットチェックなどとも呼ばれます)は、ひんぱんに必要になります。シューティングゲームでは、プレイヤーと敵、プレイヤーの弾と敵、プレイヤーと敵の弾など、さまざまな種類の衝突検出が行われます。さらに、それらが1秒間に30回の移動が行われるとすると、そのぶんだけ衝突検出が必要となります。

バウンディングボックス

ゲームのようにリアルタイム性を追求する場合、キャラクターの複雑な形状を完全に考慮した衝突検出をするのはたいへんです。実際には、バウンディングボックス(Bounding Box:衝突範囲)と呼ばれる架空の領域を用いて、その領域の交差判定を行います。

2次元の場合には、矩形や円を用いることが多く、この領域ともう一方のキャラクターの交差判定を行うことで、擬似的に衝突を検出します。たいいていの場合、キャラクターをすっぽりと包むような矩形にします。そうでない場合には、キャラクターのめり込みが起こる可能性が出てきます。



バウンディングボックス

上の図のバウンディングボックスの頂点Aの座標を(Ax, Ay)、頂点Cを(Cx, Cy)、頂点A'を(A'x, A'y)、頂点C'を(C'x, C'y)と表します。このとき、2つのバウンディングボックスが交差しているかどうかを調べるプログラムは、次のようになります。

```
if(Ax <= C'x && Cx >= A'x && Ay <= C'y && Cy >= A'y)
    交差している
else
    交差していない
```

スクリーン座標系のy軸は、下方向に向いていることに注意しましょう。条件式をまとめると、

- ・頂点Aが頂点C'の左側にあり、かつ、頂点Cが頂点A'の右側にある
- ・頂点Aが頂点C'の上側にあり、かつ、頂点Cが頂点A'の下側にある

となります。この2つの条件を満たしたとき、2つの領域は交差しています。この条件を言い換えれば、四角形A B C Dの一部または全部が、四角形A' B' C' D'の中にあるかどうかということです。

逆に言えば、

- ・頂点Aが頂点C'の右側にある
- ・頂点Cが頂点A'の左側にある
- ・頂点Aが頂点C'の下側にある
- ・頂点Cが頂点A'の上側にある

の条件を1つでも満たすと、絶対に交差していないということになります。これをJavaおよびC言語で記述すると、次のようなプログラムになります。

```
// 衝突判定([ax1, ay1]が頂点A、[ax2, ay2]が頂点C、[bx1, by1]が頂点A'、[bx2, by2]が頂点C')
if(ax2 < bx1 || ax1 > bx2 || ay1 > by2 || ay2 < by1)
    // 衝突していない
else
    // 衝突している
```

課題

衝突を判定するメソッドを作成しましょう。

(1) 矩形の交差を判定するisIntersectメソッドを作成します。以下のプログラムを適切な場所に追加しましょう。

```
//--- 交差判定
boolean isIntersect(int ax1, int ay1, int ax2, int ay2, int bx1, int by1, int bx2, int by2) {
    if(ax2 < bx1 || ax1 > bx2 || ay1 > by2 || ay2 < by1)
        return false; // 衝突していない
    return true; // 衝突している
}
```

(2) キャラクターの衝突を検出し、衝突した場合のプログラムを記述するcollisionDetectionメソッドを作成します。以下のプログラムを適切な場所に追加しましょう。

```
//--- 衝突検出
void collisionDetection() {
}
```

(3) 内部処理で衝突検出を行うようにします。procメソッドを以下のように変更しましょう。

```
//--- 内部処理
void proc() {
    player(); // プレイヤー処理
    enemy(); // 敵処理
    collisionDetection(); // 衝突検出
}
```

この変更により、「プレイヤーの処理(playerメソッド) 敵の処理(enemyメソッド) 衝突検出(collisionDetectionメソッド)」という流れになり、キャラクターの移動後、衝突の検出ができるようになります。

(4) プレイヤーと敵の衝突を検出し、衝突した場合に「あたり」とコンソールに出力されるようにしてみます。collisionDetectionメソッドを以下のように変更しましょう。

```
//--- 衝突検出
void collisionDetection() {
    // プレイヤーと敵の衝突判定
    if(isIntersect(plyX + 0, plyY + 0, plyX + 42, plyY + 55,
                  enmX + 0, enmY + 0, enmX + 113, enmY + 168)) {
        System.out.println("あたり");
    }
}
```

プレイヤーと敵にバウンディングボックスを設定し、それが交差しているかどうかをisIntersectメソッドで調べます。バウンディングボックスは、以下のようにキャラクターを完全に覆うようになっているので、画像が存在しない場所も衝突しているとみなされます。

(enmX + 0, enmY + 0)



(enmX + 113, enmY + 168)

左上の頂点(enmX + 0, enmY + 0)の数値を増やすと、領域が右下に狭まります。同様に、右下の頂点(enmX + 113, enmY + 168)の数値を減らすと、頂点が左上に狭まります。

たとえば、左上の頂点を(enmX + 37, enmY + 34)、右下の頂点を(enmX + 50, enmY + 50)とすると、顔の部分だけが衝突領域になります。

課題

プレイヤーと敵にライフを設定し、画面上部に表示しましょう。また、スコアも追加して表示しましょう。

(1) ライフは、変数で管理します。プレイヤーと敵のライフを格納する変数を宣言します。変数plyHPとenmHPを以下のように宣言しましょう。

```
// プレイヤー
MediaImage  plyMi;
Image       plyImg;
int         plyX, plyY;
int         plyAnm;
int         plyHP; // HP

// 敵
MediaImage  enmMi;
Image       enmImg;
int         enmX, enmY;
int         enmHP; // HP
```

(2) initメソッドでライフの初期値を設定します。initメソッドを以下のように変更しましょう。

```
//--- 初期化
void init() {
    // 変数初期化
    plyX   = 0;
    plyY   = 0;
    plyAnm = 0;
    plyHP  = 5;

    enmX   = 127;
    enmY   = 0;
    enmHP  = 10;
    enmMvCnt = 0;

    // 画像読み込み
```

(3) drawメソッドでライフを描画するようにします。drawメソッドを以下のように変更しましょう。

```
//--- 描画
void draw() {
    grp.lock(); // 描画開始

    grp.drawImage(bgImg, 0, 0); // 背景描画
    grp.drawImage(enmImg, enmX, enmY); // 敵描画
    grp.drawImage(plyImg[plyAnm], plyX, plyY); // プレイヤー描画

    // HP描画
    grp.setColor(Graphics.getColorOfRGB(255, 255, 255));
    grp.drawString("PLAYER " + plyHP, 0, 10);
    grp.drawString("ENEMY " + enmHP, 180, 10);

    grp.unlock(true); // 描画終了
}
```

drawStringメソッドは、文字を描画するメソッドです。描画の前にsetColorメソッドで色を設定しています。その中のGraphics.getColorOfRGBメソッドが色を作成するメソッドです。左から光の三原色の赤、緑、青の明るさで、これらが混合されて色が作成されます。色は、0～255で指定します。大きいほど明るくなります(黒からそれぞれの成分に近づきます)。

(4) スコアを格納する変数を宣言します。以下のプログラムを適切な場所に追加しましょう。

```
int score;
```

(5) initメソッドでスコアを初期化します。initメソッドの適切な場所に、以下のプログラムを追加しましょう。

```
score = 0;
```

(6) drawメソッドでスコアを描画するようにします。drawメソッドの適切な場所に、以下のプログラムを追加しましょう。

```
grp.drawString("SCORE " + score, 90, 10);
```

課題

プレイヤーまたは敵のライフが0以下になったらゲームが終了するようにしましょう。また、そのときに「おしまい」の画像を表示するようにしましょう。

(1) 以下の方法(アルゴリズム)で作成します。

1. ゲームが進行している状態と、ゲームが終了した状態(ゲームオーバー)の2つの状態が存在することになります。2つの状態によって、処理を分岐させます。
2. ゲームの状態は、変数で管理します。
3. 2の変数が1なら通常状態、2ならゲームオーバーとします。
4. ゲームオーバーなら「おしまい」と表示します。そのとき、ゲームは進行させません。

(2) ゲームオーバーかどうかを示す変数を宣言します。変数gameOverを以下のように宣言しましょう。

```
// 敵動きデータ
String      enmMove = "";
int         enmMvCnt;    // 動きデータカウンタ

int         score;

// ゲームモード
int         gameMode;
```

変数gameModeには、通常状態なら「1」、ゲームオーバーなら「2」を設定します。ゲームを拡張する場合、0ならタイトル、3ならエンディングというようにしていきます。

(3) initメソッドで変数gameModeを初期化します。ゲームを起動したときは、通常状態から始まるようにします。initメソッドを以下のように変更しましょう。なお、「?」は各自考えてください。

```
// --- 初期化
void init() {
    // 変数初期化
    gameMode = ?;

    plyX    = 0;
    plyY    = 0;
```

(4) 「おしまい」画像を読み込みます。画像を格納する変数osiImgを以下のように宣言しましょう。

```
// ゲームオーバー
int         gameMode;
MediaImage osiMi;
Image      osiImg;
```

(5) 画像を読み込みます。initメソッドの適切な場所に、以下のプログラムを追加しましょう。

```
// おしまい
osiMi = MediaManager.getImage("resource:///OSHI.gif");
osiMi.use();
osiImg = osiMi.getImage();
```

(6)内部処理をゲームの状態によって分岐させます。ゲームオーバーの場合、「おしまい」と表示するだけなので、内部処理を行う必要はありません。ゲームオーバーでない場合は、これまでどおりプレイヤー、敵、衝突検出を行います。

内部処理procメソッドを以下のように変更しましょう。なお、「?」は各自考えてください。

```
//--- 内部処理
void proc() {
    if(gameMode == ?)
        return;
    player();           // プレイヤー処理
    enemy();           // 敵処理
    collisionDetection(); // 衝突検出
}
```

この変更により、ゲームオーバーの場合は内部処理が実行されなくなります。

(7)ゲームオーバーの場合、「おしまい」と表示するようにします。drawメソッドを以下のように変更しましょう。なお、「?」は各自考えてください。

```
//--- 描画
void draw() {
    grp.lock();           // 描画開始

    grp.drawImage(bgImg, 0, 0);           // 背景描画
    grp.drawImage(enmImg, enmX, enmY);    // 敵描画
    grp.drawImage(plyImg[plyAnm], plyX, plyY); // プレイヤー描画

    // HP、スコア描画
    grp.setColor(Graphics.getColorOfRGB(255, 255, 255));
    grp.drawString("PLAYER " + plyHP, 0, 10);
    grp.drawString("ENEMY " + enmHP, 180, 10);
    grp.drawString("SCORE " + score, 90, 10);

    // おしまい描画
    if(gameMode == ?)
        grp.drawImage(osiImg, 42, 102);

    grp.unlock(true);           // 描画終了
}
```

(8)プレイヤーまたは敵のライフが0以下になったら、状態をゲームオーバーに変更します。この処理は、内部処理の最後に行います。procメソッドを以下のように変更しましょう。

```
//--- 内部処理
void proc() {
    if(gameMode == ?)           // この?は、(6)の問題になっています
        return;

    player();           // プレイヤー処理
    enemy();           // 敵処理
    collisionDetection(); // 衝突検出

    // ゲームオーバー判定
    if(plyHP <= 0 || enmHP <= 0)
        gameMode = 2;
}
```

追加した判定により、どちらかのライフが0以下になると変数gameModeが2になり、ゲームオーバー状態となります。この状態になると、(6)により内部処理が実行されなくなり(=ゲームが進行しない)、(7)により「おしまい」と表示されるようになります。

(9)プレイヤーと敵が衝突したら、プレイヤーのライフを0にします。以下のプログラムを適切な場所に追加しましょう。なお、「System.out.println("あたり")」は削除してかまいません。

```
plyHP = 0;
```

(10)プレイヤーと敵が衝突した場合、プレイヤーのライフが0になってゲームオーバーになるかどうか確認しましょう。

プレイヤーが弾を発射できるようにしましょう。

(1) 以下の方法(アルゴリズム)で作成します。

- ・携帯電話の真ん中のボタン(SELECTキー)が押されたときに弾を発射
- ・発射できるのは1発のみ
- ・言い換えれば、画面に弾がないとき発射可能
- ・発射された弾は右に移動する
- ・画面の端または敵に当たると消滅(=未発射状態となる)
- ・敵に当たるとダメージ1(敵のライフが1減る)、スコア1,000点加算
- ・弾は、「発射されている(画面内に存在する)状態」と「発射されていない(画面内に存在しない)状態」という2つの状態が存在する
- ・弾の状態は、変数で管理

(2) 弾を管理する変数を宣言します。必要な変数は以下のものです。

- ・画像(pshMi:MediaImage型、pshImg:Image型)
- ・発射されているかどうか(pshLNC:boolean型...trueは発射中、falseは未発射)
- ・x座標(pshX:int型)
- ・y座標(pshY:int型)

以上の変数を宣言する以下のプログラムを適切な場所に追加に追加しましょう。

```
// プレイヤーショット
MediaImage  pshMi;
Image       pshImg;
boolean     pshLNC;
int         pshX;
int         pshY;
```

(3) 変数を初期化します。initメソッドの適切な場所に、以下のプログラムを追加しましょう。

```
pshLNC = false;
```

弾の初期化は、発射状態をfalseにして未発射状態にしておくだけで十分です。座標は発射したときに、プレイヤーの座標をもとに決定します。

(4) 画像を読み込みます。initメソッドの適切な場所に、以下のプログラムを追加しましょう。

```
// プレイヤーショット
pshMi = MediaManager.getImage("resource:///SHT00.gif");
pshMi.use();
pshImg = pshMi.getImage();
```

(5) 弾が未発射状態でSELECTキーが押されたら、弾を発射するようにします。この処理は、プレイヤーに関する処理なので、playerメソッドに記述します。

以下のプログラムをplayerメソッドのに追加しましょう。なお、「?」は各自考えてください。

```
// ショット発射
if(pshLNC == false) {
    if((1 << Display.KEY_?????? & key) != 0) { // 発射されていないか調べる
        if(SELECTキーが押されているか調べる) { // SELECTキーが押されているか調べる
            pshLNC = true; // 発射状態にする
            pshX = plyX + 36; // 座標の設定(杖の先)
            pshY = plyY + 22;
        }
    }
}
```

弾が発射されていない状態(変数pshLNCがfalse)のとき、SELECTキーが押されたら、弾を発射状態(変数pshLNCをtrue)にし、弾の座標をプレイヤーの座標plyX, plyYをもとに変数pshX, pshYに設定します。

(6) 弾を描画するようにします。弾を描画するのは、発射されているときだけです。発射されていないときに描画してはいけません。

以下のプログラムをdrawメソッドに追加しましょう。

```
grp.drawImage(plyImg[plyAnm], plyX, plyY); // プレイヤー描画
// プレイヤーショット描画
if(pshLNC == true)
    grp.drawImage(pshImg, pshX, pshY);

// HP、スコア描画
grp.setColor(Graphics.getColorOfRGB(255, 255, 255));
```

弾を描画するのは、発射されているとき(変数pshLNCがtrueのとき)だけなので、描画の前に判定を行い、条件を満たすときだけ描画します。

(7)とりあえず、弾を発射できるようになったので、プログラムを実行して確認してみましょう。

(8)弾を移動させる処理を作成します。必要な処理は、以下のものです。

- ・弾を移動させる
- ・弾が完全に画面外に出たら弾を未発射状態にする
- ・ただし、これらの処理を行うのは、弾が発射されているときだけ

この処理は、プレイヤーの弾を処理する専用のメソッドplayerShotを作成し、そこに記述します。以下のプログラムの「？」を埋め、適切な場所に追加しましょう。

```
//--- プレイヤーショット処理
void playerShot() {
    if(pshLNC == ????) { // 弾が発射されているか調べる
        // 弾の移動
        pshX += 4;

        // 弾が画面内にあるか調べる
        // (弾の領域と画面の領域が交差しているか調べる = 交差していなければ、画面外にある)
        if(isIntersect(pshX, pshY, pshX + 24, pshY + 22,
                      0, 0, 240, 240) == false)
            pshLNC = ?????; // 未発射状態にする
    }
}
```

(9)内部処理procメソッドで、弾の処理が行われるようにします。procメソッドを以下のように変更しましょう。

```
//--- 内部処理
void proc() {
    if(gameMode == 2)
        return;

    player(); // プレイヤー処理
    enemy(); // 敵処理
    playerShot(); // プレイヤーショット処理
    collisionDetection(); // 衝突検出

    // ゲームオーバー判定
    if(plyHP <= 0 || enmHP <= 0)
        gameMode = 2;
}
```

(10)以上で弾が移動するようになったので、プログラムを実行して確認してみましょう。

(11)弾と敵の衝突検出を行います。衝突した場合、敵のライフを1減らし、スコアに1,000を加算します。以下のプログラムの「？」を埋め、collisionDetectionメソッドの適切な場所に追加しましょう。

```
//--- 衝突検出
void collisionDetection() {
    // プレイヤーショットと敵の衝突検出
    if(pshLNC == ????) { // 発射されているときだけ判定する
        if(isIntersect(pshX, pshY, pshX + 24, pshY + 22,
                       enmX + 37, enmY + 34, enmX + 50, enmY + 50)) {
            pshLNC = ?????; // 弾を消す
            enmHP -= 1; // 敵のHPを減らす
            score += 1000; // スコアに1000を加算
        }
    }
}

// プレイヤーと敵の衝突判定
```

プレイヤーの弾と敵の衝突検出も、弾が発射されているときだけ行います。弾は、敵の領域を調整し、顔の部分だけにしか当たらないようにしています。

衝突した場合、弾を消し(状態を未発射にする)、敵のライフから1を引く、スコアに1000を加算します。

課題

敵が弾を発射するようにしましょう。

(1)以下の方法(アルゴリズム)で作成します。

- ・発射できるのは1発のみ
- ・言い換えれば、画面に弾がないとき発射可能
- ・発射する確率は、条件を満たしたときに10%の確率
- ・発射された弾は左に移動する
- ・画面の端またはプレイヤーに当たると消滅(=未発射状態となる)
- ・プレイヤーが当たるとダメージ1(プレイヤーのライフが1減る)
- ・プレイヤーの弾と敵の弾が当たった場合は、両方とも消滅し、スコアに100を加算
- ・弾は、「発射されている(画面内に存在する)状態」と「発射されていない(画面内に存在しない)状態」という2つの状態が存在する
- ・弾の状態は、変数で管理

(2)以下のプログラムを適切な場所に追加しましょう。なお、「？」は各自考えてください。

- ・乱数を扱うための変数の宣言

```
Random rand = new Random();
```

「import java.util.Random;」が必要

- ・敵の弾を管理する変数の宣言

```
// 敵ショット
MediaImage eshMi;
Image eshImg; // 画像
boolean eshLNC; // 発射状態
int eshX; // x座標
int eshY; // y座標
```

- ・敵の弾を初期化するプログラム

```
eshLNC = false;
```

- ・敵の弾の画像を読み込むプログラム

```
// 敵ショット
eshMi = MediaManager.getImage("resource:///ENMSHT.gif");
eshMi.use();
eshImg = eshMi.getImage();
```

・敵が弾を発射するプログラム

```
// ショット発射
if(eshLNC == ????) { // 発射されていない
    if(Math.abs(rand.nextInt() % (100 / 10)) == 0) { // 10%の確率
        eshLNC = true; // 発射状態にする
        eshX = enmX + 24; // 座標の設定(口のあたり)
        eshY = enmY + 32;
    }
}
```

敵の弾が未発射のとき、0～9のランダムな数値を作成し、それが0のとき(=1/10の確率)に弾を発射します。なお、この処理は敵に関する処理なので、enemyメソッドに記述します。

・敵の弾を描画するプログラム

```
// 敵ショット描画
if(eshLNC == ????)
    grp.drawImage(eshImg, eshX, eshY);
```

プレイヤーの弾と同様に、発射されているときだけ描画します。

・敵の弾を移動するプログラム

```
//--- 敵ショット処理
void enemyShot() {
    if(eshLNC == ????) {
        // 弾の移動
        eshX -= 6;

        // 弾が画面内にあるか調べる
        // (弾の領域と画面の領域が交差しているか調べる = 交差していなければ、画面外にある)
        if(!isIntersect(eshX, eshY, eshX + 18, eshY + 32, 0, 0, 240, 240)) eshLNC = ????. // 未発射状態にする
    }
}
```

・内部処理procメソッドへ追加

```
enemyShot(); // 敵ショット処理
```

・プレイヤーとの衝突を判定するプログラム

```
// プレイヤーと敵の弾の衝突判定
if(eshLNC == ????) { // 発射されているときだけ判定する
    if(isIntersect(plyX + 0, plyY + 0, plyX + 42, plyY + 55,
        eshX + 0, eshY + 0, eshX + 18, eshY + 32)) {
        eshLNC = ????. // 弾を消す
        plyHP -= 1; // プレイヤーのHPを減らす
    }
}
```

・プレイヤーの弾との衝突を判定するプログラム

```
// プレイヤーの弾と敵の弾の衝突検出
if(pshLNC == ??? && eshLNC == ???) { // 両方が発射されているときだけ判定する
    if(isIntersect(pshX, pshY, pshX + 24, pshY + 22,
        eshX, eshY, eshX + 18, eshY + 32)) {
        pshLNC = ????. // 弾を消す
        eshLNC = ????.
        score += 100; // スコアに100を加算
    }
}
```

課題

プレイヤーをアニメーションさせましょう。

(1) プレイヤーの「画像を保存する変数」を配列にします。変数plyMiとplyImgの宣言を以下のように変更しましょう。

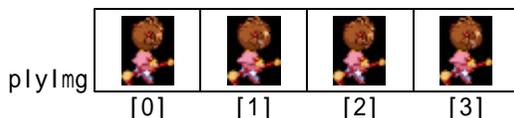
```
// プレイヤー
MediaImage[] plyMi = new MediaImage[4];
Image[]      plyImg = new Image[4];
int          plyX, plyY;
```

4コマ分の画像があるので、要素数4の配列を作成します。

(2) 配列に画像を読み込みます。initメソッドの「// 画像読み込み」のプレイヤーの部分を以下のように変更しましょう。

```
// プレイヤー
for(int i = 0; i < plyMi.length; i++) {
    plyMi [i] = MediaManager.getImage("resource:///PLY0" + i + ".gif");
    plyMi [i].use();
    plyImg[i] = plyMi[i].getImage();
}
```

for文で配列plyMiの要素の数(plyMi.length)だけ繰り返して画像を読み込みます。要素数は4なので、ループカウンタiが0～3の間for文の内部が実行されます。プレイヤーのファイル名はPLY00.gif～PLY03.gifで、ちょうど"PLY0"とi(0～3)と".gif"を足した文字列と同等になります。これを利用して効率よくファイルを読み込んでいます。



(3) 「何コマ目の画像を描画するのか」を保存する変数plyAnmを宣言します。「// プレイヤー」を以下のように変更しましょう。

```
// プレイヤー
MediaImage[] plyMi = new MediaImage[4];
Image[]      plyImg = new Image[4];
int          plyX, plyY;
int          plyAnm;
```

(4) initメソッドで変数plyAnmを初期化します。initメソッドを以下のように変更しましょう。

```
// 変数初期化
plyX   = 0;
plyY   = 0;
plyAnm = 0;
;
```

(5) 描画するコマを計算します。描画するコマは4コマなので、変数plyAnmを0から3まで増やし、3を超えたら0に戻します。playerメソッドを以下のように変更しましょう。

```
// --- プレイヤー処理
void player() {
    // キー入力取得
    int key = getKeypadState();

    // 移動
    (省略)

    // 移動限界処理
    (省略)

    // アニメーション処理
    plyAnm = (plyAnm + 1) % plyImg.length;
}
```

変数plyAnmを1ずつ増やし、要素数plyImg.length(今回は4)で割った余りを求めることにより、0から3の値しかとらなくなります(plyAnmが4になると0になります)。

(6)プレイヤーの描画をアニメーション対応にします。以下のように変更しましょう。

```
grp.drawImage(plyImg[plyAnm], plyX, plyY); // プレイヤー描画
```

配列plyImgの添字に変数plyAnmを用いることにより、plyImg[0]～plyImg[3]を順に描画していきます。

i アプリのダウンロード

エミュレータでひと通りの動作確認ができれば、実機で最終確認を行います。電話がかかってきて、i アプリの動作が中断したときなど、エミュレータでは完全に再現できない部分もあり、また実機での動作速度など細かいところは微妙に異なるからです。

実機で動かすには、i アプリを実行するのに必要なファイルを携帯電話機に転送しなければなりません。PCから携帯電話機に直接転送することはできません。Web(http)サーバを用意し、i アプリをダウンロードできるようにする必要があります。

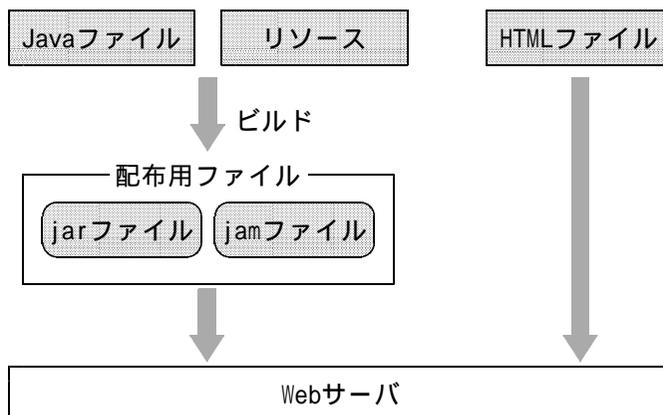
i アプリをビルドしたときに作成されるjamファイル(i アプリの情報が記述されているファイル)とjarファイル(複数のクラスファイルやリソースファイルをひとつにまとめたファイル)をWebサーバにftなどで転送します。

次に、これらのファイルを携帯電話機にダウンロードできるように、以下のようなHTMLファイルを作成します。

```
<HTML>
<BODY>
<OBJECT declare id="application" data="ファイル名.jam" type="application/x-jam">
</OBJECT>
<A ijam="#application" href="iAppliNotSupported.html">ダウンロード</a>
</BODY>
</HTML>
```

<OBJECT>タグで指定したidと<A>タグで指定した「ijam="#"に続く文字列(上の例ではapplication)が一致している必要があります。また、<A>タグの「href="」に続く文字列(上の例ではiAppliNotSupported.html)は、i アプリ対応の携帯電話以外の機種やPC等から閲覧したときに、「ダウンロード」のリンクを選択したときに表示させるページへのリンクです。このページには、「i アプリがサポートされていません」などのメッセージを表示させるとよいでしょう。

上記のHTMLファイルもjamファイルとjarファイルがあるのと同じ場所に転送します。そして、HTMLファイルのURLを携帯電話機で指定すると、「ダウンロード」というリンクが表示されるので、[選択]キーを押してダウンロードします。ダウンロードが終われば、電話機を操作してi アプリを起動することができます。



課題

i アプリを携帯電話機にダウンロードし、実機で動作確認しましょう。