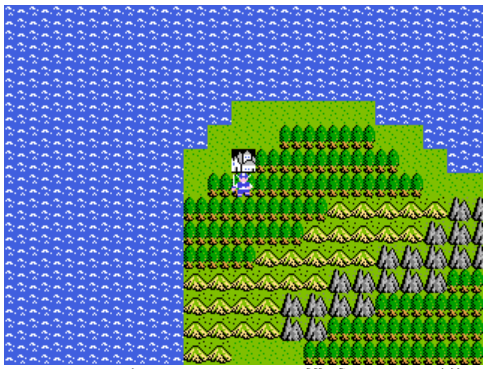
オブジェクト指向とゲームプログラミング

§　アルゴリズム編 － 第６回　タイルグラフィックス　§

タイルグラフィックス

タイルグラフィックスとは、右のような８×８、16×16ドットといった矩形の画像を複数作成し、それを並べることによって作られるグラフィックスのことです。タイルを並べるようにグラフィックスが描画されるため、タイルグラフィックスと呼ばれます。容量がほぼ無限にあるPCゲームではあまり使われませんが、ROMカートリッジや携帯電話のような容量の制限が厳しいゲームで用いられています。

ビットマップの場合、640×480ドットで画面を構成すると、１画面あたり256色の場合は約300キロバイト、フルカラー(24ビット)の場合は約900キロバイトの容量が必要になります。これに対し、タイルグラフィックスでは、小さな画像を並べるので、かなり容量が少なくなります。どのぐらいの容量になるかは、タイルの数で変わります。さらに、タイルの並べ方を変えるだけで、さまざまな画像を作成することができます。



▲タイルグラフィックスで構成された画像

課　　題

タイルグラフィックスを用いて画像を描画してみましょう。

(１)タイルを作成します。１つのタイルは16×16ドットとします。このようなタイルを８パターン以上、なるべくたくさん作成しましょう。

(２)画像のパターン(タイルの並び方)を定義します。

タイルをどのように並べるかは、２次元配列で定義します。このとき、各タイルに番号を割り当てておく必要があります。番号は、計算しやすいように、０から始まる連番の数字(または何らかの法則性のある数列)で定義します。今回は、左上のタイルが０で、そのとなりが１、さらにそのとなりが２というように、右に進むたびに１増えていくというルールにします。

(１)の画像では、８×３個のタイルが並んでいます。この場合は、左上の黒いタイルが０、そのとなりの平地が１、森が２を割り当てます。このとき、１つ下の段のバリアは８、城は９となります。つまり、となりは１、下は８の差があることになります。この法則は非常に重要です。

0	1	2	3	4	5	6	7
8	9	…			…	14	15
16	17	…			…	22	23
⋮	⋮					⋮	⋮

タイルに数字を割り当てたので、２次元配列でタイルをどのように並べるかを定義します。描画時に、この配列で指定されたタイルを並べ、画像を構築します。
以下のプログラムを適切な場所に追加しましょう。

```
// タイルのサイズ
public final static int   MAP_CHIP = 16;

// マップデータ
//                    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
int[][]   map = {{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 0
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 1
                 {1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 2
                 {1, 1, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 3
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 4
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 5
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 6
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 7
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 8
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 9
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 10
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 11
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 12
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},     // 13
                 {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}};    // 14
```

※マップデータは、手作業で作成するのは非常に困難です。マップエディタ(例：http://www.valley.ne.jp/~valencia/)があると、GUIを用いてタイルを配置し、そのデータを出力することができます。

(３)タイル画像を読み込むのに必要な変数を定義します。以下のプログラムを適切な場所に追加しましょう。

```
// マップ
MediaImage    mapMi;
Image         mapImg;
```

(４)タイル画像を読み込むプログラムを作成し、適切な場所に追加しましょう。

(５)２次元配列をもとにタイルを並べ、画像を構築するプログラムを作成します。drawメソッドの適切な場所に、以下のプログラムを完成させて追加しましょう。

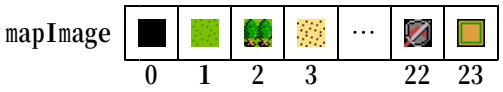
```
// マップ構築
for(int y = 0; y < ※１; y++) {
    for(int x = 0; x < ※２; x++) {
        grp.drawScaledImage(mapImg,
                            ※３,
                            ※４,
                            ※５,
                            ※６,
                            map[y][x] ※７ MAP_CHIP,
                            map[y][x] ※８ MAP_CHIP,
                            ※９,
                            ※10);
    }
}
```

ヒント：
※１　ｙ方向(縦)ループの継続条件が入ります。縦に、いくつタイルを並べるかです。
※２　ｘ方向(横)ループの継続条件が入ります。横に、いくつタイルを並べるかです。
※３　転送先(画面)のｘ座標。１ブロックずつ転送するので、繰り返されるたびにタイルの幅の分だけずらしていきます。
※４　転送先(画面)のｙ座標。１ブロックずつ転送するので、繰り返されるたびにタイルの高さの分だけずらしていきます。

※５　転送先の領域の幅。転送元領域の幅と異なる場合、拡大縮小されます。
※６　転送先の領域の高さ。転送元領域の幅と異なる場合、拡大縮小されます。
※７　転送元(タイル)のｘ座標。２次元配列をもとに計算します。今回の例では、タイルは８パターン並んでいるので、配列の値が０，８，１６，２４の場合は「０」、１，９，１７，２５の場合は「１」、２，１０，１８，２６の場合は「２」にタイルの幅を掛けた値となります。
※８　転送元(タイル)のｙ座標。２次元配列をもとに計算します。今回の例では、タイルは８パターン並んでいるので、配列の値が０から７の場合は「０」、８から１５の場合は「１」、１６から２３の場合は「２」にタイルの幅を掛けた値となります。
※９　転送元の領域の幅。転送先領域の幅と異なる場合、拡大縮小されます。
※10　転送元の領域の高さ。転送先領域の高さと異なる場合、拡大縮小されます。
※※　たとえば、配列から取り出した値が「１２」の場合、宝箱のタイルを描画することになります。このとき、転送元のｘ座標は６４，転送元のｙ座標は１６となります。

map[y][x]の値が１２の場合…

(６)タイルを１パーツごとに分割して配列に読み込むと、配列の値をタイル配列の添え字に使用できるので、(５)の計算式が不要になります。



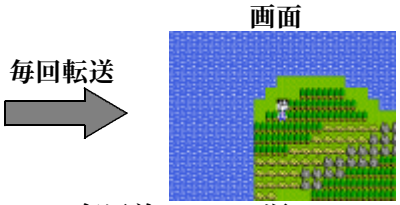

```
// マップ構築
for(int y = 0; y < ※１; y++) {
    for(int x = 0; x < ※２; x++) {
        grp.drawImage(mapImg[map[y][x]], ※３, ※４);
    }
}
```

(７)タイルを並べた画像を保存しておく場所(キャッシュ)を作成し、描画速度を向上させましょう。

描画のたびにタイルを並べて画像を構築すると、転送回数が多いので動作は遅くなります。スクロールするにしても、変更のある部分はわずかなので、毎回画像を構築する必要はありません。そこで、タイルを並べた画像を保存しておく場所(キャッシュ)を作成しておき、画像はそこに構築します。画面に描画する際は、キャッシュから転送します。スクロールなどで画像に変化が起きた場合は、キャッシュの画像を変更します。このようにすれば、毎回タイルを並べる必要がないので、転送回数が減り、動作速度はかなり向上します(ただし、転送するピクセル数は同じなので、向上しない場合もあります)。

・キャッシュなし

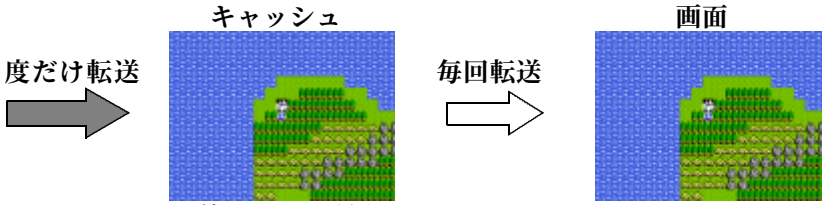
```
// マップデータ
int[][]   map = {{1, 1, 1, …
                 {1, 1, 1, …
                 {1, 1, 2, …
```



毎回並べるので遅い…

・キャッシュなし

```
// マップデータ
int[][]   map = {{1, 1, 1, …
                 {1, 1, 1, …
                 {1, 1, 2, …
```



変更があったときだけ並べるので遅くならない

以下のプログラムを適切な場所に追加しましょう。

－変数の追加－

```
// マップキャッシュ
Image       mapCacheImg;
Graphics    mapCacheGrp;
```

－マップキャッシュの生成(initメソッドに追加)－

```
// マップキャッシュ生成
mapCacheImg = Image.createImage(Display.getWidth(), Display.getHeight());
mapCacheGrp = mapCacheImg.getGraphics();

// マップキャッシュに画像を構築
for(int y = 0; y < Display.getHeight() / MAP_CHIP; y++) {
    for(int x = 0; x < Display.getWidth() / MAP_CHIP; x++) {
        mapCacheGrp.drawScaledImage(mapImg,
                                    ※3,
                                    ※4,
                                    ※5,
                                    ※6,
                                    map[y][x] ※7 MAP_CHIP,
                                    map[y][x] ※8 MAP_CHIP,
                                    ※9,
                                    ※10);
    }
}
```

－マップキャッシュの描画(drawメソッドに追加)－

```
grp.drawImage(mapCacheImg, 0, 0);
```

※スクロールする場合は、現在の画像をスクロール方向にずらし、新たに出現する部分だけマップキャッシュに描画します。

応用問題　DoJa3.5から、２次元配列をもとにタイルを並べて画像を構築するImageMapが導入されました。上記のプログラムを、ImageMapを使ったものに変更しましょう。