陳昇瑋
（原名：陳寬達）

姓名標示-非商業性-相同方式分享 2.5

您可自由：

- 重製、散布、展示及演出本著作
- 創作衍生著作

惟需遵照下列條件：

姓名標示. 您必須按照作者或授權人所指定的方式，保留其姓名標示。

非商業性. 您不得爲商業目的而使用本著作。

相同方式分享. 若您改變、轉變或改作本著作，僅在遵守與本著作相同的授權條款下，您始得散布由本著作而生的衍生著作。

- 爲再使用或散布本著作，您必須向他人清楚說明本著作所適用的授權條款。
- 如果您取得著作權人之許可，這些條件中任一項都能被免除。

您合理使用的權利及其他的權利，不因上述內容而受影響。

這是一份讓一般人易於了解的法律條款（完整的授權條款）摘要。

免責聲明

第八章

足球番

許多人應該都很熟悉「倉庫番」這個小遊戲，

在各個平臺及各式電視遊樂器上皆曾見它的蹤跡。

我的回味方式比較奇特，不是好好地玩玩它，

而是以 Delphi 從頭到尾撰寫一套足球版的「足球番」。

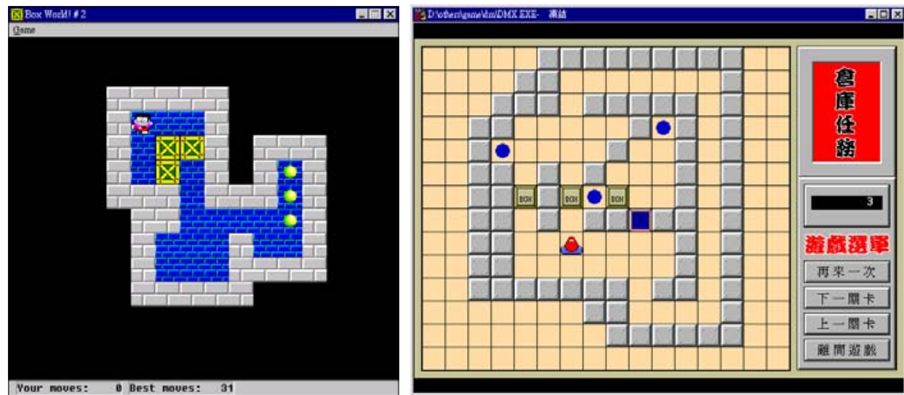

圖 8-1 / 兩個從 Internet 上取得的倉庫番遊戲

上圖是兩個從 Internet 上取得的倉庫番遊戲，皆是國人的作品。前者似乎以純 Windows SDK 開發，後者以 QBasic 4.5 英文版撰寫。

你一定也很熟悉這個小遊戲吧！在各個平臺及各式電視遊樂器上皆曾見它的蹤跡。規則十分簡單，只要操縱主角將箱子一一推至地圖上的標示點，就算過關。雖然規則就這麼簡單，但難度可不低唷，在細心安排設計下，有些關卡總要讓人傷透腦筋，嘗試個十來次甚至百來次才能過關[1]。

這個再簡單也不過的老遊戲，正是今天的主題。我們將以 Delphi 從頭到尾撰寫一套足球版的「倉庫番」，就稱它為「足球番」好了。它的特性十分符合本章的教學目的：

1. 畫面處理 GDI 即可輕鬆解決，背景不用捲動，也不算即時遊戲。
2. 除遊戲主程式外，必須另有地圖／關卡編輯器的搭配才算完整，另外還配有圖庫編輯器，這兩支工具可推廣使用於許多益智、角色扮演甚至動作遊戲上頭。
3. 圖形使用不多，但是遊戲本身耐玩，不是須靠畫面才能吸引人的遊戲類型。

先來訂立我們這套足球番的功能及特色：

---

[1] 好吧，我承認，上頭那兩個遊戲我就有些關卡過不了。:P

1. 遊戲規則與倉庫番皆相同，將所有目的地形利用覆蓋用物品掩住即可過關。
2. 地圖大小即是可視範圍，因此不用支援地圖捲動。
3. 所有圖片，包括角色圖形尺寸大小皆相同。
4. 角色的移動以圖片大小為單位，因此沒有小碎步移動所帶來的碰撞處理問題。
5. 所有圖片，包括角色圖形皆是外掛方式，可以在不修改程式碼的情形下更動圖形。
6. 採用關卡制度，可供使用者自行編輯關卡。
7. 具有動作重播功能。

看起來，除了有些新鮮有點無聊的重播功能外，只不過是另一套簡單的倉庫番類型遊戲，連圖形、角色移動等方式都還一切從簡哩。希望你不會太失望，功夫從易處練，雖然一點都不炫，至少是自己寫出來的嘛。

系統規劃

全套遊戲除了主程式外，另設計兩支工具程式－圖庫編輯器及地圖編輯器。因為重覆性高，有些遊戲將地圖編輯器及主程式合併放在同一支程式內，但是，分開為兩支程式有下列優點：

- 減少程式設計的複雜度。同一支程式提供的功能越多，程式邏輯必定越複雜。
- 不必要的 overhead。地圖編輯不一定提供給 end user，若將這些功能置於主程式中而無法使用，徒然增加檔案大小而已。

好，那表示我們要分別撰寫三支應用程式了。在動手之前，別急，讓我們先將系統必要的幾個重要類別切割出來。

TTiles 類別

用來定義一個「圖庫」（Tile Archive），一個圖庫包含多張圖片，畫面上除了角色圖形外的所有圖片都是由圖庫中取得。

因爲提供給「地形」及「物品」兩層地圖的圖片屬性完全不同，因此必須爲這兩個圖庫分別設計不同的圖庫類別，所以我將 *TTiles* 設定爲抽象類別，但做好大部分的工作，如載入／儲存圖庫，圖片管理及繪製圖片等等，再分別由 *TTerrTiles* 及 *TItemTiles* 兩個類別繼承，分別提供設定及讀取圖片屬性的功能。

- 「地形」圖片有「可以通過」及「目的地形」兩種屬性。
- 「物品」圖片有「可以移動」及「覆蓋用物品」兩種屬性。

TMap 類別

用來定義一張地圖，或說，一個關卡的佈局。本遊戲的地圖設計爲兩層，底下一層是地形，上面一層是物品，兩層獨立操作／貼圖互不相干，貼圖用的圖片分別由兩個圖庫提供，因此分別需要一個二維陣列來儲存圖片編號。

爲求視覺逼真效果及操作編輯方便，現在的地圖往往都不只分爲兩層，如 StarCraft 就分爲五層（請見圖 8-2），英雄無敵 III 也至少分爲四層（請見圖 8-3），分別是土地、河流、道路及地形物。我們的足球番由於地圖簡單，因此分爲地形及物品兩層即足夠。

TMap 類別不但負責關卡的載入及儲存，另外也會儲存角色的初始位置。

圖 8-2 / StarCraft 的地圖編輯器，它有五層地形可供編輯。

圖 8-3 / 英雄無敵 III 的地圖編輯器，它有四層地形可供編輯。

TRole 類別

角色類別，負責角色的圖形載入及繪製，移動本身的位置計算及搬動物品。另外重播功也由此類別自行紀錄移動資訊，再交由主程式來播放。

總共才需五個類別，而且除了 *TTiles* 類別是抽象類別，不用來產生物件外，其它四個類別在遊戲中都只要產生一個物件就夠了，爲什麼？因爲我們同一時間只使用一張地圖，一個「地形」圖庫，一個「物件」圖庫及一個角色。

類別實作

先將類別的功能及需求定義出來，切割清楚後，一一將這些重要類別實作出來，到時候用「兜」的，便可輕易兜出三個程式來了，信不信？:）

首先得先定義一些到處用得著的常數（定義於 Util.pas）：

```
const
  TILE_WIDTH        = 32; // 圖片寬度點數
  TILE_HEIGHT       = 32; // 圖片高度點數

  TILE_NUM_X        = 10; // 畫面橫軸格數
  TILE_NUM_Y        = 10; // 畫面縱軸格數

  FN_TERR_ARCHIVE   = 'TERR.TIA'; // 地形圖庫
  FN_ITEM_ARCHIVE   = 'ITEM.TIA'; // 物品圖庫
  FN_ROLEBITS       = 'ROLEBITS.BMP'; // 角色圖檔

  FN_MAP_PREFIX     = 'MAP'; // 關卡圖檔檔名 (MAP??.DAT)
  FN_MAP_EXT        = '.DAT'; // 關卡圖檔副檔名

  SIG_MYFILE        = 'Xshadow_Stock'; // 圖庫及地圖檔案的檔頭標籤
```

圖片大小爲 32 x 32，是十分常見且有效率的大小設定，因爲我們的 CPU 通用暫存器寬度也是 32 bit，在進行記憶體區塊搬移時，不會有不符合 DWORD alignment 的情況發生。

畫面橫軸及縱軸格數是隨手定義的，在我的 1280 x 1024 解析度畫面下看起來小小的，但後來才發現 10 x 10 格設計不出複雜的關卡。:P 反正只要將這兒的常數更動，重新編譯主程式及地圖編輯器後，即可以新的畫面大小進行關卡設計及遊戲，覺得不夠的讀者請自己修改。:p

SIG_MYFILE 用來作爲檔頭標籤，在讀取圖庫及地圖檔案時，先確認檔案開頭有沒有此字串，以確定讀取的是我們自己的檔案，不會有誤讀的情況發生。更甚者，我在圖庫的檔頭標籤後頭又加上「副檔頭標籤」，用以辨別、確認「地形」及「物品」圖庫。

TTiles 圖庫類別及子類別

TTiles 是第一個實作的類別，這是它的類別宣告（定義於 Tiles.pas）：

```
#0001  type
#0002    TTiles = class
#0003    private
#0004      FBits: TBitmap; // 存放圖庫所有圖片的 bitmap
#0005
#0006      FTileNum: Integer; // 圖片數量
#0007
#0008      function GetTilePos_Left(index: Integer): Integer;
#0009      function GetTilePos_Top(index: Integer): Integer;
#0010
#0011      function GetTileNumPerRow: Integer;
#0012      function GetTileRowCount: Integer;
#0013    protected
#0014      procedure SetTileNum(Value: Integer); virtual;
#0015
#0016      // 讀取及設定屬性都是虛擬方法，讓後代類別來改寫
#0017      procedure LoadAttrs(fs: TFileStream); virtual; abstract;
#0018      procedure WriteAttrs(fs: TFileStream); virtual; abstract;
#0019    public
#0020      constructor Create;
#0021      destructor Destroy; override;
#0022
#0023      // 載入及儲存圖庫
#0024      procedure LoadFromFile(Filename: string);
#0025      procedure SaveToFile(Filename: string);
#0026
```

```
#0027     procedure ImportPicture(Filename: string); // 匯入圖形檔
#0028
#0029     property TileNum: Integer read FTileNum write SetTileNum;
#0030
#0031     // 每列的圖片數目
#0032     property TileNumPerRow: Integer read GetTileNumPerRow;
#0033
#0034     // 共有幾列圖片
#0036     property TileRowCount: Integer read GetTileRowCount;
#0037
#0038     property Bitmap: TBitmap read FBits;
#0039
#0040     // 根據圖片編號就可取得圖片在圖形中的位置
#0041     property TilePos_Left[index: Integer]: Integer read
#0042       GetTilePos_Left;
#0043     property TilePos_Top[index: Integer]: Integer read
#0044       GetTilePos_Top;
#0045   end;
#0046
#0047 const
#0048   TILE_BITS_NUM_X = 10; // 圖庫中每列圖片的數目
```

取巧的圖片儲存方式

圖庫的實作採用了一個很偷懶很取巧的方法：將所有的圖片擺在同一個 bitmap 中，同時定義了一個 *TILE_BITS_NUM_X* 常數，表示圖庫中每列圖片的數目。我將此常數設定爲 10，表示圖庫 bitmap 爲寬 = 320 點，高 = 圖片數目 / 10 x 32，所以編號爲 0 .. 9 的圖片會依序擺在第一列、編號爲 10 .. 19 的圖片會擺在第二列等等。

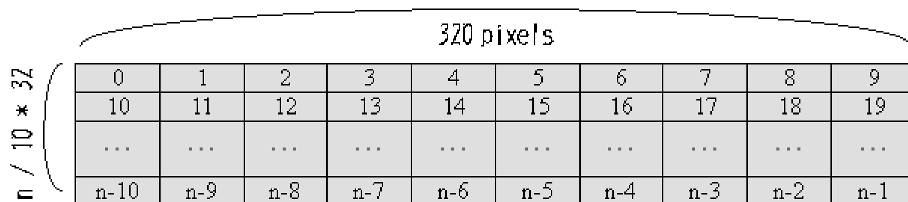

圖 8-4 / 圖庫中圖片的儲存方式

這種偷懶的圖片儲存方法讓設計者在畫好圖片後必須開啓影像編輯軟體，手動將圖片擺

在正確的位置上，麻煩極了。例如，我畫好 No.13 圖片，這時必須打開原本儲存圖庫的圖形檔，將它擺在 （96, 32） 的位置上，差一點都不行。呵，我知道你們都會罵我很笨，饒了我吧，下不為例，為了程式簡單，我只好出此下策，下回我便會將它改成比較聰明的儲存方式。

為了這種圖片儲存法，*TTiles* 類別提供 *TilePos_Left* 及 *TilePos_Top* 兩個陣列型態屬性，來讓外界依圖片編號就可取得圖片在圖檔中的 X 軸及 Y 軸位置。這兩個屬性皆屬於一維陣列屬性，使用時必須傳入索引值，而它們對應的 *GetTilePos_Left* 及 *GetTilePos_Top* 兩個函式才會根據此索引值去計算圖片位置。

另外圖片數目也一定為 *TILE_BITS_NUM_X* 的倍數，是呼叫 *ImportPicture* 方法後根據此圖形檔的長寬大小自動計算而得，*ImportPicture* 方法同時會裁切載入的圖形檔，使圖形檔的寬度變成 *TILE_WIDTH * TILE_BITS_NUM_X*，高度變成 *TILE_HEIGHT* 的整數倍：

```
#0001  procedure TTiles.ImportPicture(Filename: string);
#0002  begin
#0003    FBits.LoadFromFile(Filename);
#0004    TileNum := FBits.Height div TILE_HEIGHT * TILE_BITS_NUM_X;
#0005
#0006    // 將 bitmap 裁減為所需的大小
#0007    FBits.Width := TILE_WIDTH * TILE_BITS_NUM_X;
#0008    FBits.Height := TileNum div TILE_BITS_NUM_X * TILE_HEIGHT;
#0009  end;
```

載入圖檔的 *TTiles.LoadFromFile* 方法使用 VCL 的 *TFileStream* 類別來開啟檔案及讀取資料，0018 行還呼叫了 *LoadAttrs* 方法供後代類別 *TTerrTiles* 及 *TItemTiles* 來改寫，讀取它們自己的屬性資料：

```
#0001  procedure TTiles.LoadFromFile(Filename: string);
#0002  var
#0003    fs : TFileStream;
#0004    Val: Integer;
#0005  begin
#0006    fs := TFileStream.Create(Filename, fmOpenRead);
#0007    try
#0008      CheckSignature(fs, SIG_MYFILE); // 檢查檔頭標籤
#0009      CheckSignature(fs, ClassName); // 檢查副檔頭標籤
#0010
#0011      fs.Read(Val, sizeof(Integer)); // 讀入圖片數目
```

```
#0012     if Val = 0 then
#0013       raise Exception.Create('No tiles');
#0014     TileNum := Val; // 爲了觸發 property 的 SetXXX method
#0015
#0016     // 這是虛擬方法，因爲 TTiles 本身根本沒有定義屬性，
#0017     // 讓後代類別決定該如何讀取屬性
#0018     LoadAttrs(fs);
#0019
#0020     FBits.LoadFromStream(fs); // 讀入存放所有圖片的圖形
#0021
#0022     // 檢查圖形的長寬符不符合標準
#0023     if FBits.Width < TILE_WIDTH * TILE_BITS_NUM_X then
#0024       raise Exception.Create('Width of tile bitmap is invalid');
#0025
#0026     if FBits.Height div TILE_HEIGHT * TILE_BITS_NUM_X < FTileNum
#0027       then raise Exception.Create('Height of bitmap is invalid');
#0028   finally
#0029     fs.Free;
#0030   end;
#0031 end;
```

你可以在類別宣告中注意到，*LoadAttrs* 及 *WriteAttrs* 兩個方法不但爲虛擬方法，同時還宣告爲抽象方法，這是因爲 *TTiles* 類別根本沒有圖片屬性的概念，只是宣告好這兩個方法，完全不實作，留待後代改寫使用。

地形及物品圖庫類別

TTiles 類別撰寫完成後，接著就可以從它衍生出談論已久，呼之欲出的 *TTerrTiles* 及 *TItemTiles* 類別了（與 *TTiles* 類別同樣定義於 Tiles.pas）。

```
#0001 type
#0002   TTerrAttr = set of (taCanPass, taTarget); // 地形的屬性
#0003   TItemAttr = set of (iaCanMove, iaSource); // 物品的屬性
#0004
#0005   TTerrTiles = class(TTiles)
#0006   private
#0007     FAttrs: array of TTerrAttr;
#0008     function GetAttrs(index: Integer): TTerrAttr;
#0009     procedure SetAttrs(index: Integer; const Value: TTerrAttr);
#0010   protected
#0011     procedure SetTileNum(Value: Integer); override;
```

```
#0012
#0013     procedure LoadAttrs(fs: TFileStream); override;
#0014     procedure WriteAttrs(fs: TFileStream); override;
#0015   public
#0016     property Attrs[index: Integer]: TTerrAttr read GetAttrs write
#0017       SetAttrs;
#0018   end;
#0019
#0020   TItemTiles = class(TTiles)
#0021   private
#0022     FAttrs: array of TItemAttr;
#0023     function GetAttrs(index: Integer): TItemAttr;
#0024     procedure SetAttrs(index: Integer; const Value: TItemAttr);
#0025   protected
#0026     procedure SetTileNum(Value: Integer); override;
#0027
#0028     procedure LoadAttrs(fs: TFileStream); override;
#0029     procedure WriteAttrs(fs: TFileStream); override;
#0030   public
#0031     property Attrs[index: Integer]: TItemAttr read GetAttrs write
#0032       SetAttrs;
#0033   end;
#0034
#0035 var
#0036   Terrs: TTerrTiles;
#0037   Items: TItemTiles;
```

第 0002、0003 行是這兩個新類別的重點，兩個圖庫類別分別擁有不同的圖片屬性。0007 及 0022 分別宣告兩個類別用來儲存屬性的動態陣列，你可以注意它沒有像一般的 Pascal array 宣告方式標明陣列的上下註標，這是 Delphi 4 才提供的新特性。在宣告此動態陣列後，可以隨時利用 *SetLength* 及 *Length*、*Min*、*Max* 等函式來設定及取得此動態陣列的長度及上下註標。

兩個類別都改寫了 *TTiles* 類別的 *SetTileNum* 及 *LoadAttrs*、*WriteAttrs* 方法，目的十分簡單，改寫 *SetTileNum* 方法是為了在圖片數目改變時同時變更 *FAttrs* 屬性陣列的長度，而 *LoadAttrs*／*WriteAttrs* 則經由 *TFileStream* 物件來讀取及寫入屬性陣列。以 *TTerrTiles* 類別的程式碼為例：

```
procedure TTerrTiles.SetTileNum(Value: Integer);
begin
  inherited SetTileNum(Value);
  SetLength(FAttrs, FTileNum);
```

```
end;

procedure TTerrTiles.LoadAttrs(fs: TFileStream);
begin
  fs.Read(FAttrs[0], sizeof(TTerrAttr) * FTileNum);
end;

procedure TTerrTiles.WriteAttrs(fs: TFileStream);
begin
  fs.Write(FAttrs[0], sizeof(TTerrAttr) * FTileNum);
end;
```

0036 及 0037 列分別為 *TTerrTiles* 及 *TItemTiles* 類別各宣告一個物件，因為它們只使用一個物件便已足夠，因此在此宣告，在三支程式中使用時就不用分別再為它們宣告了。

好了，完成了三個類別，我將它們置於 Tiles 單元。

TMap 地圖類別

嘿，接下來輪到 *TMap* 類別了（好像在宰羊圈裏的待宰綿羊似的:p），*TMap* 類別定義於 MapUnit.pas：

```
#0001  type
#0002    TMapArray = array[0..TILE_NUM_Y - 1, 0..TILE_NUM_X - 1] of Byte;
#0003
#0004    TMap = class
#0005    private
#0006      FTerrMap, FItemMap: TMapArray; // 地形及物品地圖
#0007
#0008      FLevelNo: Integer; // 目前載入的關卡編號
#0009      FInvBitmap: TBitmap; // 用於物品的透明貼圖
#0010
#0011      FRole_X, FRole_Y: Integer; // 角色的起始位置
#0012
#0013      procedure SetLevelNo(const Value: Integer);
#0014
#0015      function GetCanPass(X, Y: Integer): Boolean;
#0016      function GetIsTarget(X, Y: Integer): Boolean;
#0017
#0018      function GetCanMove(X, Y: Integer): Boolean;
#0019      function GetIsSource(X, Y: Integer): Boolean;
```

```
#0020
#0021     function GetTerrAttr(X, Y: Integer): TTerrAttr;
#0022     procedure SetTerrAttr(X, Y: Integer; const Value: TTerrAttr);
#0023     function GetItemAttr(X, Y: Integer): TItemAttr;
#0024     procedure SetItemAttr(X, Y: Integer; const Value: TItemAttr);
#0025
#0026     function GetTerrMap(X, Y: Integer): Byte;
#0027     function GetItemMap(X, Y: Integer): Byte;
#0028     procedure SetTerrMap(X, Y: Integer; const Value: Byte);
#0029     procedure SetItemMap(X, Y: Integer; const Value: Byte);
#0030     procedure SetRole_X(const Value: Integer);
#0031     procedure SetRole_Y(const Value: Integer);
#0032   protected
#0033   public
#0034     constructor Create;
#0035     destructor Destroy; override;
#0036
#0037     function GetFileName: string; // 根據關卡編號，傳回對應的檔名
#0038
#0039     procedure LoadFromFile; // 讀入地圖檔
#0040     procedure SaveToFile; // 儲存地圖檔
#0041
#0042     procedure DrawTerrMap(Canvas: TCanvas); // 將地形畫在 Canvas 上
#0043     procedure DrawItemMap(Canvas: TCanvas); // 將物品畫在 Canvas 上
#0044
#0045     // 重設整張地圖，或只重設地形或物品
#0046     procedure ResetMap;
#0047     procedure ResetTerrs;
#0048     procedure ResetItems;
#0049
#0050     property LevelNo: Integer read FLevelNo write SetLevelNo;
#0051
#0052     // 取得某格的地形及物品圖片編號
#0053     property TerrMap[X, Y: Integer]: Byte read GetTerrMap write
#0054       SetTerrMap;
#0055     property ItemMap[X, Y: Integer]: Byte read GetItemMap write
#0056       SetItemMap;
#0057
#0058     // 取得初始的角色位置
#0059     property Role_X: Integer read FRole_X write SetRole_X;
#0060     property Role_Y: Integer read FRole_Y write SetRole_Y;
#0061
#0062     // 取得某格的地形屬性
#0063     property TerrAttr[X, Y: Integer]: TTerrAttr read GetTerrAttr
#0064       write SetTerrAttr;
#0065
```

```
#0066     property CanPass[X, Y: Integer]: Boolean read GetCanPass;
#0067     property IsTarget[X, Y: Integer]: Boolean read GetIsTarget;
#0068
#0069     // 取得某格的物品屬性
#0070     property ItemAttr[X, Y: Integer]: TItemAttr read GetItemAttr
#0071       write SetItemAttr;
#0072
#0073     property CanMove[X, Y: Integer]: Boolean read GetCanMove;
#0074     property IsSource[X, Y: Integer]: Boolean read GetIsSource;
#0075   end;
#0076
#0077 var
#0078   Map: TMap;
```

光是類別宣告就洋洋灑灑的七十二行，其實絕大部分是屬性宣告及對應的讀取／寫入方法，讓 *TMap* 類別使用起來更方便罷了。

首先宣告 *TMapArray* 型別，它是 *TILE_NUM_X * TILE_NUM_Y* 個元素的 *Byte* 陣列，*Byte* 型態的範圍爲 0 ~ 255，表示圖片最多只能有 256 種，我知道這時又有人想 K 我了，不急，換個角度想，現在限制越多，表示改良空間越大，以後得進步獎的機會也越高說。不過 256 個圖片對於足球番這種小遊戲來講委實也夠多了，不夠的話隨時再將 *Byte* 改爲 *Word* 甚至 *Cardinal*（相當於 C/C++ 語言的 *unsigned int* 型態，四位元組的無號正整數）也行。

0006 列宣告 *FTerrMap* 及 *FItemMap* 兩個型態爲 *TMapArray* 的地形及物品地圖。在這兒我做了特殊的設定：**「0 號地形爲預設地形，而 0 號物品表示此處無物品」**。因爲地形一定遍布整張地圖，但物品不是，所以一定得設定一個數字表示此處沒有東西，而 0 號是最佳選擇。

處理透明貼圖

FInvBitmap 物件是專門用來供物品圖片做透明貼圖用的。透明貼圖指的是將物品「貼」到背景上時，周圍不屬於物品本身之處，就應該讓背景顯現出來，如圖 8-5。

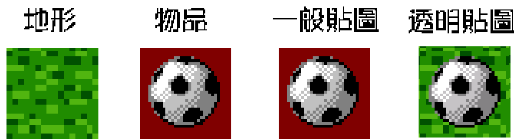

圖 8-5 / 一般貼圖及透明貼圖的比較

透明貼圖一直是 GDI 繪圖中十分傷腦筋的問題。從前 DOS 時代在 X mode 下，連貼圖動作都是自己寫的，利用兩層迴圈將像素「塞」進顯示卡的視訊記憶體，完全掌握著最低階的動作，因此達成透明貼圖的效果只是舉手之勞。而 DirectDraw 的 *ISurface* 也提供了 *SetColorKey* 等函式提供透明貼圖的解決方案。

但在 GDI 中要達成透明貼圖的效果可就麻煩了，一來我們對它的掌控能力不若 DOS 下直接填寫視訊記憶體那麼完整，二來 GDI 本身動作就慢，不容許我們花太多額外時間在處理透明貼圖。於是，最早最早，在 Windows 95 那個時代，GDI API 還沒有提供任何關於透明貼圖的解決方案前，達成透明貼圖通常必須經由下列的九大步驟：

實驗器材：

原始影像一，遮罩顏色一，遊戲畫面一。

實驗目的：

將原始影像貼到遊戲畫面上，但不貼原始影像中遮罩顏色的部分，使得那一部分仍然呈現原本的遊戲畫面。

實驗步驟：

1. 建立一個 DC 來放置原始影像，稱之為「影像 DC」。
2. 將原始影像選擇至 DC 上。
3. 另外建立一個 memory DC 來存放最後的影像，暫且稱它為「目的 DC」。
4. 將畫面上將要貼圖的矩形區域圖形拷貝到目的 DC。
5. 建立一個「AND 遮罩」，讓原始影像所有以遮罩顏色繪製的部分通通變成透明的，建立「AND 遮罩」有下列三小步驟：

- 將「影像 DC」的背景顏色設定為遮罩顏色。
- 建立一個相同大小的單色 DC。
- 將原始影像不管三七二十一貼到此單色 DC 上。這使得此單色 DC 變成原始影像透明貼圖用的「AND 遮罩」，此遮罩於原始影像為遮罩顏色部分呈現 1，而非遮罩顏色部分呈現 0。

6. 呼叫 *BitBlt* 函式，搭配 *SRCAND* 貼圖模式將「AND 遮罩」貼到「目的 DC」上。
7. 呼叫 *BitBlt* 函式，搭配 *SRCAND* 貼圖模式將經過反相的「AND 遮罩」貼到「影像 DC」上。
8. 呼叫 *BitBlt* 函式，搭配 *SRCPAINT* 貼圖模式將「影像 DC」貼到「目的 DC」上。
9. 最後將「影像 DC」貼到畫面上適當的位置。

呼，看得都很累了，這是 MSDN 裏建議使用的方法，當然不是唯一做法囉，只是方法大同小異，簡單不到哪去。

Tips

你可以發現這九道步驟完全沒有涉及調色盤的處理，所以這方法只適用於不使用調色盤的顯示模式，不適合 256 色或 16 色模式使用。

但是，若你的程式只想在 Windows NT 上執行，Windows NT 提供一道新的 *MaskBlt* 函式，讓我們可以很簡單地達成透明貼圖，只是這一點都不實用，有誰想要寫只能在 Windows NT 執行的遊戲呢？

雖然微軟知錯能改，亡羊補牢，早已提出 *TransparentBlt*、*AlphaBlend* 透明及半透明貼圖等 GDI 函式，但只在 Windows 2000 才提供，遠水救不了近火，我們還是自求多福，自己動手做透明貼圖囉。

很幸運地，VCL 的 *TCanvas* 及 *TBitmap* 類別提供透明貼圖的機制，只要將 *TBitmap* 的 *Transparent* 屬性設為 *True*，將 *TransparentColor* 設為遮罩顏色，再呼叫 *TCanvas.Draw* 方法就行了：

```
#0001  procedure TForm1.btnVCLClick(Sender: TObject);
#0002  var
#0003    Bits: TBitmap;
#0004  begin
#0005    Bits := TBitmap.Create; // 建立暫時的 TBitmap
#0006    Bits.Assign(imgSrc.Picture.Bitmap); // 將原始影像拷貝過來
#0007    Bits.Transparent := True;
#0008    Bits.TransparentColor := RGB(128, 0, 0); // 設定遮罩顏色
#0009
#0010    imgDst.Canvas.Brush.Color := clBtnFace;
#0011    imgDst.Canvas.FillRect(imgDst.Canvas.ClipRect);
#0012    imgDst.Canvas.Draw(0, 0, Bits); // 複製到目的畫布上
#0013    Bits.Free;
#0014  end;
```

要注意的是，*TBitmap* 的這幾個 *TransparentXXXX* 屬性只針對 *TCanvas* 的 *Draw* 及 *BrushCopy* 方法有效，對於 *TCanvas* 的其它方法或 GDI 函式皆無效用，所以若你將 0012 行改成 *BitBlt* API 函式或 *TCanvas* 的 *StretchBlt* 方法時，會發現完全沒有透明貼圖的效果。你可以在 Graphics 單元中找到實作透明貼圖的 *TransparentStretchBlt* 函式，若在 Windows NT 下而且原始及目的影像大小一樣時，它就直接呼叫 *MaskBlt* GDI 函式；對於其它版本的 Windows，就仿前頭的九大步驟，先製作「AND 遮罩」，經過幾道邏輯貼圖，再加上調色盤的處理，以支援 256 色或 16 色模式下的透明貼圖效果。

Info

十分遺憾的是，*TCanvas* 的透明貼圖能力在 Windows 98 竟然出問題，無論 Delphi 4 + Update 3 或最近的 Delphi 5，這個問題依舊存在，Borland 真該打。

真是氣煞人也，沒關係，幸好我們有那九陽真經..哦，不，是透明貼圖九步心訣，大不了自己做一個就是，沒什麼了不起。我寫了一個小範例程式，分別使用 VCL 及 API 來達成透明貼圖，執行結果請見下頁圖 8-6。

由於九大步驟程式碼寫起來拉里拉砸地好長一串，我就不貼出來了，順便在程式中也附上 C 語言版本的寫法，可供與 Object Pascal 版本對照閱讀。

圖 8-6 / 分別使用 VCL 及 API 來達成透明貼圖的範例程式

Okay，回到正題，我們提到，*FInvBitmap* 物件是專門用來供物品圖片做透明貼圖用的，因此在 *TMap* 的建構方法中，可以看到 *FInvBitmap* 的屬性設定：

```
constructor TMap.Create;
begin
  inherited Create;

  ...

  FInvBitmap := TBitmap.Create;
  FInvBitmap.Width := TILE_WIDTH;
  FInvBitmap.Height := TILE_HEIGHT;
  FInvBitmap.Transparent := True;
  FInvBitmap.TransparentColor := RGB(128, 0, 0);
end;
```

待會便拿它來達成貼物品圖片的透明貼圖效果。而遮罩顏色 *RGB(128, 0, 0)* 是我任意選定的，但選定就不要更動，因爲遊戲中所有的物品及角色圖片都必須嚴格遵照這個遮罩顏色來繪製，若有更動便要大肆修改，十分麻煩。

關卡檔案的載入儲存

TMap 類別的載入及儲存方法會分別將兩層地圖，及初始角色位置放到檔案或從檔案讀出。這裏的改進空間極大，也可將其它與關卡相關的設定一併儲存，例如關卡描述啦、過關提示啦、過關美女圖啦，都一起放到關卡檔案中。*TMap.LoadFromFile* 方法如下：

```
#0001  procedure TMap.LoadFromFile;
#0002  var
#0003    fs: TFileStream;
#0004  begin
#0005    fs := TFileStream.Create(GetFileName, fmOpenRead);
#0006    with fs do
#0007      try
#0008        CheckSignature(fs, SIG_MYFILE);
#0009        CheckSignature(fs, ClassName);
#0010
#0011        Read(FRole_X, sizeof(Integer)); // 讀取角色座標
#0012        Read(FRole_Y, sizeof(Integer));
#0013
#0014        Read(FTerrMap, sizeof(TMapArray)); // 讀取地形地圖
#0015        Read(FItemMap, sizeof(TMapArray)); // 讀取物品地圖
#0016      finally
#0017        Free;
#0018      end;
#0019  end;
```

0008 及 0009 列同樣地檢查檔頭標籤及副檔頭標籤，接下來讀取角色位置，最後才是兩張固定大小的地形及物品地圖，過程平鋪直述，十分簡單直覺。

TMap 類別的重頭戲是 *DrawTerrMap* 及 *DrawItemMap* 兩個分別繪製地形層及物品層畫面的方法，裏頭同樣地都是兩層迴圈，一一尋訪所有的圖格，根據那一格的地形或物品圖片編號，將圖片畫在 *Canvas* 上頭：

```
#0001  procedure TMap.DrawTerrMap(Canvas: TCanvas);
#0002  var
#0003    X, Y: Integer;
#0004  begin
#0005    for Y := 0 to TILE_NUM_Y - 1 do
#0006      for X := 0 to TILE_NUM_X - 1 do
#0007        BitBlt(Canvas.Handle, X * TILE_WIDTH, Y * TILE_HEIGHT,
#0008          TILE_WIDTH, TILE_HEIGHT, Terrs.Bitmap.Canvas.Handle,
#0009          Terrs.TilePos_Left[FTerrMap[Y, X]],
#0010          Terrs.TilePos_Top[FTerrMap[Y, X]], SRCCOPY);
#0011  end;
#0012
#0013  procedure TMap.DrawItemMap(Canvas: TCanvas);
#0014  var
#0015    X, Y     : Integer;
#0016    Left, Top: Integer;
#0017  begin
```

```
#0018   // 統一 FInvBitmap 及 Items.Bitmap 的格式
#0019   FInvBitmap.PixelFormat := Items.Bitmap.PixelFormat;
#0020
#0021   for Y := 0 to TILE_NUM_Y - 1 do
#0022     for X := 0 to TILE_NUM_X - 1 do
#0023       if (FItemMap[Y, X] <> 0) then
#0024       begin
#0025         Left := Items.TilePos_Left[FItemMap[Y, X]];
#0026         Top := Items.TilePos_Top[FItemMap[Y, X]];
#0027
#0028         FInvBitmap.Canvas.CopyRect(Rect(0, 0, TILE_WIDTH,
#0029           TILE_HEIGHT), Items.Bitmap.Canvas,
#0030           Rect(Left, Top, Left + TILE_WIDTH, Top + TILE_HEIGHT));
#0031
#0032         // 解決 VCL 在 Windows 98 下透明貼圖 bug 的 workaround
#0033         if IsWindows98 then
#0034           TransparentBlt(Canvas.Handle, FInvBitmap.Handle, X *
#0035             TILE_WIDTH, Y * TILE_HEIGHT, RGB(128, 0, 0))
#0036         else
#0037           Canvas.Draw(X * TILE_WIDTH, Y * TILE_HEIGHT, FInvBiTMap);
#0038       end;
#0039 end;
```

0007 列繪製地形層比較單純，呼叫 *BitBlt* 函式，將對應的地形圖片拷貝到 *Canvas* 上的對應位置。但繪製物品層時，首先要檢查該格是否有物品，若 *FItemMap[Y, X]* = 0 表示沒有物品，就不畫；另外要繪製時，先將圖片拷貝至 *FInvBitmap* 上，再呼叫 *Canvas.Draw* 方法貼上 *FInvBitmap*，目的就是為了先前討論已久的透明貼圖。理論上是如此，但我們先前提過 VCL 的透明貼圖能力在 Windows 98 下失效，所以 0032 行就專為解決 VCL 在 Windows 98 下透明貼圖 bug 而特別將 Windows 98 平臺下的貼圖動作獨立出來，先以 *IsWindows98* 函式（在 Utils.pas）判斷目前是否處於 Windows 98 下，若是的話，再呼叫我們自己的 *TransparentBlt* 函式來達成透明貼圖。

大一統的 Win32 平臺？！

經常撰寫 Win32 程式的程式員，一定會對微軟吶喊的「統一的 Win32 平臺」這句好聽的口號印象深刻吶！真的只是口號，我很少寫出一套不需針對不同 Windows 版本修正問題的軟體。判斷系統是否為 Windows 98 並不難，檢查 VCL 提供的三個全域變數

Win32Platform，*Win32MajorVersion* 及 *Win32MinorVersion* 即可，詳情請查閱 *GetVersionEx* API 函式。而 Windows 98 即等於版本爲 4.10 以上的 Windows：

```
function IsWindows98: Boolean;
begin
  Result := (Win32Platform = VER_PLATFORM_WIN32_WINDOWS) and
    ((Win32MajorVersion > 4) or
    ((Win32MajorVersion = 4) and (Win32MinorVersion >= 10)));
end;
```

若系統爲 Windows NT，則 *Win32Platform* 變數值爲 *VER_PLATFORM_WIN32_NT*；可以想見即將面世的 Windows 2000，其 *Win32MajorVersion* = 5，因爲其實就是 NT 5.0 嘛。

二維陣列屬性

最後再實作 *TMap* 提供的一堆屬性，以 *CanPass* 屬性爲例，它是二維陣列屬性，宣告爲：

```
  property CanPass[X, Y: Integer]: Boolean read GetCanPass;
```

它有一個對應的屬性讀取方法，因此就要另外實作此方法：

```
function TMap.GetCanPass(X, Y: Integer): Boolean;
begin
  Result := taCanPass in Terrs.Attrs[FTerrMap[Y, X]];
end;
```

於是 *TMap* 的物件使用者就可以輕易地使用此屬性，如經由 *Map.CanPass[2, 3]* 取得一個布林值，判斷此地圖座標爲 （2, 3） 的格點是否能夠讓角色通過；由 *Map.CanMove[8, 2]* 來判斷座標爲 （8, 2） 處是否有物品，若有物品，能不能推動等等。

TMap 類別宣告的 0078 列，宣告一個 *TMap* 類別的全域物件 *Map*，原因也跟 *TTerrTiles* 及 *TItemTiles* 類別一樣，因爲 *TMap* 物件只需要一個，因此宣告在這兒最清楚也最方便。

TRole 主角類別

終於剩下最後一個類別－*TRole*（定義於 Role.pas）：

```
#0001  type
#0002    TDirection = (drUp, drDown, drLeft, drRight);
#0003
#0004    TRole = class
#0005    private
#0006      FX, FY: Integer; // 角色座標
#0007      FDirection: TDirection; // 行進方向
#0008      FBits, FInvBitmap: TBitmap; // 角色圖片及透明貼圖用圖片
#0009
#0010      FPlayBackList, FMoveList: TList; // 重播功能用的動作記錄
#0011    public
#0012      constructor Create;
#0013      destructor Destroy; virtual;
#0014
#0015      procedure LoadBits; // 載入角色的 bitmap
#0016      procedure Draw(Canvas: TCanvas); // 繪製角色
#0017      procedure Move(Dir: TDirection); // 移動角色（碰撞處理，移動物品）
#0018
#0019      procedure SavePlayback;    // 將動作記錄移至重播紀錄
#0020      procedure CleanMoveList;   // 清除動作記錄
#0021
#0022      // 位置及方向
#0023      property X: Integer read FX write FX;
#0024      property Y: Integer read FY write FY;
#0025      property Direction: TDirection read FDirection write FDirection;
#0026
#0027      property PlayBackList: TList read FPlayBackList;
#0028    end;
```

0002 列先宣告 *TDirection* 列舉型態，定義出角色可能面對及移動的方向。*FInvBitmap* 變數的目的及用法與 *TMap* 的 *FInvBitmap* 一模一樣，同時是爲了透明貼圖功能而存在的。

TRole 的設定比較偷懶，因爲畫面上只有唯一一個角色，因此圖片也只要一份，上下左右各一張圖片，總共才需一張 128 x 32 的點陣圖。若遊戲中可同時出現多種角色，每種角色又有不同的圖形及行爲模式時，*TRole* 的設計可就沒這麼簡單，通常還要衍生不同的類別來達成。一般來說，走動時的圖片都常會一個方向提供三張，分別是站立不動、提起左腳及邁開右腳，若再加上蹲姿或射擊、中彈等其它動作表情等等，所需的圖片數量還真多，同樣地，留待日後再來加強角色的功能。

讀取角色圖形的方法很簡單，呼叫 *TBitmap.LoadFromFile* 方法從 bitmap 檔案中讀出即可：

```
#0001  procedure TRole.LoadBits;
#0002  begin
#0003    // 讀取 BMP 檔，內含四個方向的圖形
#0004    FBits.LoadFromFile(FN_ROLEBITS);
#0005
#0006    if FBits.Width < TILE_WIDTH * 4 then // 四個方向
#0007      raise Exception.Create('Width of role bits is invalid');
#0008
#0009    // 統一 FInvBitmap 及 FBits 的格式
#0010    FInvBitmap.PixelFormat := FBits.PixelFormat;
#0011  end;
```

接下來是畫出角色的 *Draw* 方法，與 *TMap.DrawItemMap* 方法類似，先將對應的圖片拷貝到 *FInvBitmap* 上，再呼叫 *Canvas.Draw* 將 *FInvBitmap* 以透明貼圖的方式貼上去。這裏也同樣有解決 VCL 在 Windows 98 下透明貼圖 bug 的 workaround。

```
#0001  procedure TRole.Draw(Canvas: TCanvas);
#0002  begin
#0003    // 先將要秀出的區域拷至 FInvBitmap，再畫出 FInvBitmap
#0004    FInvBitmap.Canvas.CopyRect(Rect(0, 0, TILE_WIDTH, TILE_HEIGHT),
#0005      FBits.Canvas, Rect(Ord(FDirection) * TILE_WIDTH, 0,
#0006      (Ord(FDirection) + 1) * TILE_WIDTH, TILE_HEIGHT));
#0007
#0008    // 解決 VCL 在 Windows 98 下透明貼圖 bug 的 workaround
#0009    if IsWindows98 then
#0010      TransparentBlt(Canvas.Handle, FInvBitmap.Handle, FX *
#0011        TILE_WIDTH, FY * TILE_HEIGHT, RGB(128, 0, 0))
#0012    else
#0013      Canvas.Draw(FX * TILE_WIDTH, FY * TILE_HEIGHT, FInvBitmap);
#0014  end;
```

移動及推動

TRole 類別最重要的方法，也是與使用者操作最直接相關的，就屬 *Move* 方法了。它決定當使用者按上下左右方向鍵後，角色的移動及搬動物品的處理：

```
#0001  // small but useful function, update x & y location of role
#0002  procedure MoveAhead(var X, Y: Integer; Dir: TDirection);
#0003  begin
#0004    case Dir of
#0005      drUp: Dec(Y);
#0006      drDown: Inc(Y);
```

```
#0007     drLeft: Dec(X);
#0008     drRight: Inc(X);
#0009   end;
#0010 end;
#0011
#0012 procedure TRole.Move(Dir: TDirection);
#0013 var
#0014   X, Y, MX, MY: Integer;
#0015 begin
#0016   X := FX; Y := FY;
#0017   MoveAhead(X, Y, Dir); // 計算角色的下一位置
#0018   FDirection := Dir;
#0019
#0020   if (Y < 0) or (X < 0) or (X >= TILE_NUM_X) or (Y >= TILE_NUM_Y)
#0021     then Exit; // 超出畫面範圍了..
#0022   if not Map.CanPass[X, Y] then Exit; // 不能走的地形
#0023
#0024   if (Map.ItemMap[X, Y] <> 0) then // 如果要移動過去的位置有物品
#0025   begin
#0026     if not Map.CanMove[X, Y] then Exit; // 不能搬動耶
#0027
#0028     MX := X; MY := Y;
#0029     MoveAhead(MX, MY, Dir); // 計算物品的下一位置
#0030     if (MY < 0) or (MX < 0) or (MX >= TILE_NUM_X) or (MY >=
#0031       TILE_NUM_Y) then Exit; // 不可將物品移到範圍外
#0032
#0033     // 要搬過去的新位置上不能有東西，也必須是可以走動的地形
#0034     if (Map.ItemMap[MX, MY] <> 0) or (not Map.CanPass[MX, MY]) then
#0035       Exit;
#0036
#0037     Map.ItemMap[MX, MY] := Map.ItemMap[X, Y]; // 搬動過去
#0038     Map.ItemMap[X, Y] := 0; // 原來的地方沒有物品了
#0039   end;
#0040
#0041   // 成功走出，將移動方向記錄下來
#0042   FMoveList.Add(Pointer(Dir));
#0043
#0044   FX := X; // 更新角色位置
#0045   FY := Y;
#0046 end;
```

先設計一個小小的 *MoveAhead* 函式來輔助位置的處理，根據傳入的 *Dir* 方向，更改 X 或 Y 軸位置，雖然簡單，但很有用。

上面處理角色移動的邏輯並不難，可歸納如下：

1. 0017 列先計算下一步的位置。
2. 0020 列判斷是否超出畫面範圍了，是的話就跳離處理函式。
3. 0022 列判斷是否將走到不能穿過的地形，是的話就跳離處理函式。
4. 0024 列判斷是否將走到有物品的圖格，是的話繼續步驟 5，否則繼繼步驟 9。
5. 0026 列判斷該物品能否搬動，否的話就跳離處理函式。
6. 0030 列判斷是否會將物品搬出畫面範圍了，是的話就跳離處理函式。
7. 0034 列判斷物品的新位置上是否有東西，是否為可以穿越的地形？若沒有擺置物品且地形可以穿越就繼續，否則跳離處理函式。
8. 0037、0038 列將物品搬動到新位置。
9. 0042 列將移動方向記錄在 *FMoveList* 物件中，留待重播功能使用。
10. 0044、0045 列更新角色位置，大功告成。

程式乍看之下可能不怎麼懂，但若用文字敘述出來，就變成很簡單的幾道判斷敘述而已，而這已是整個遊戲中最麻煩重要的邏輯處理呢。所以我說的沒錯吧，撰寫遊戲一點都不難，難的通常是顯示技術、速度及程式複雜度，而非程式邏輯。讓我們繼續往下看。

TRole 類別宣告的 0010 列中，將重播動作記錄用的兩個變數 *FPlayBackList* 及 *FMoveList* 宣告為 *TList* 類別，若你對它不熟悉，可將它看成 VCL 提供的「動態指標陣列類別」。它提供的幾個方法，如 *Add*、*Insert*、*Delete*、*Exchange*、*Clear* 等等，供我們將指標加入、插入、刪除、交換及清除這個陣列的元素，由它自動幫我們維護所需的記憶體空間，是 VCL 最常用的幾個工具類別之一。

在這兒，我想要將角色的每一步移動方向都記錄下來，但是角色的移動步數未知，可能只有兩三步，也可能是兩三千步，所以不適合靜態地配置記憶體空間。但自己呼叫 *AllocMem*、*ReAllocMem*、*FreeMem* 等函式來管理移動記錄所需的記憶體空間又稍嫌麻煩，於是 *TList* 物件就成了最適宜的容器。雖然 *TList* 類別只能處理 Pointer 型態變數，但在 Win32 平臺中，Pointer 佔用四個位元組，與整數佔用的空間一樣，所以可以放心地將任何序數型態（如整數，字元，列舉型態）等變數轉型後，存入 *TList* 物件中，如同上頭

的 0042 列。

角色動作錄影支援

TRole 的 *SavePlayback* 及 *CleanMoveList* 方法分別將 *FMoveList* 的記錄移至 *FPlaybackList* 中，以及將 *FMoveList* 的內容清除：

```
#0001  procedure TRole.SavePlayback;
#0002  var
#0003    I: Integer;
#0004  begin
#0005    FPlayBackList.Clear; // 將行動記錄放到 FPlayBackList 中, 以便重播
#0006    for I := 0 to FMoveList.Count - 1 do
#0007      FPlayBackList.Add(FMoveList[I]);
#0008  end;
#0009
#0010  procedure TRole.CleanMoveList;
#0011  begin
#0012    FMoveList.Clear; // 行動記錄清除以方便下次使用
#0013  end;
```

不知不覺間，我們已將 *TTiles*、*TMap* 及 *TRole* 等三個核心類別實作完成，好的開始是成功的一半，緊接著，就要利用上頭介紹的這些類別來架構遊戲的三支程式囉。

圖庫編輯器、地圖編輯器，以及遊戲主程式的實作也有一定的順序。圖庫編輯器還未完成前，沒有圖庫，就無法編輯地圖；而地圖編輯器還未完成前，沒有地圖，就無法進行遊戲。所以看來非從圖庫編輯器下手不可。

圖庫編輯器

RAD 開發工具的好處是，可以在開始撰寫第一行程式碼前，先透過「所視即所得」的整合環境編輯方式，將使用者介面，所有的控制項，元件，選單及視覺佈局擺好在視窗上，待程式一執行，嘿，就是設計時期擺放的那個模樣。

我個人十分偏愛這樣的設計方式，先將使用者介面設計好，再下手來撰寫程式碼。只要介面制訂後不再更動，程式碼就好寫；若是介面甚至元件種類還變來變去，那程式碼肯定就得改來改去，越修越複雜，麻煩的不得了。

好，選取【New / Application】開啓一個新專案順便建立 main form，在 form 上將介面佈局擺好，如圖 8-7，看起來很陽春，我知道，是我的錯，以後改進。圖 8-8 是其選單設計畫面，可由其得知圖庫編輯器準備提供的功能。

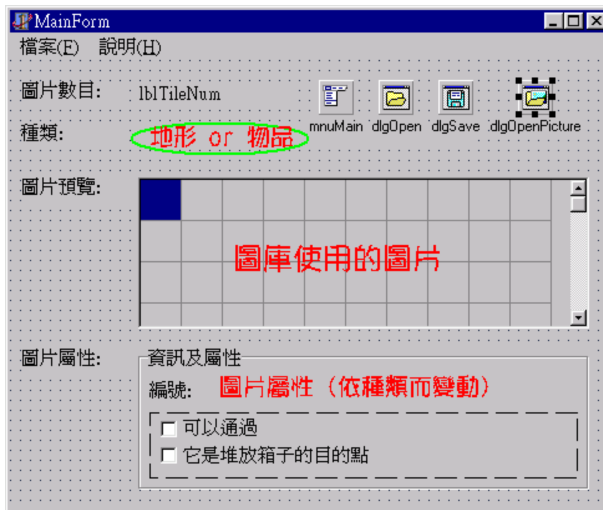

圖 8-7 / 圖庫編輯器的設計畫面

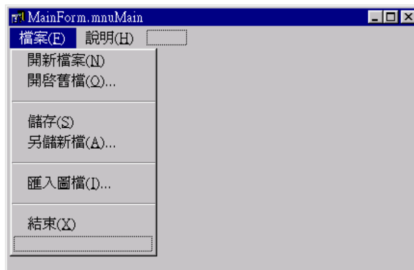

圖 8-8 / 圖庫編輯器的選單設計畫面

假設我準備好 16 張地形圖片（大小皆是 32 x 32），想要納入遊戲使用。典型的操作方式是這樣的：

1. 請先在影像編輯軟體中編輯 320 x 64（因為每列 10 格，16 張需佔用兩列）大小的影像，分別將 16 張圖片放到適當的位置，第一張擺在 （0, 0）、第二張擺在 （32, 0）、第三張擺在 （64, 0） ...，最後儲存為 BMP 檔案。
2. 接著執行圖庫編輯器，選擇【檔案 / 開新檔案】，圖庫類型選擇為「地形」，建立新的圖庫。
3. 匯入事先準備好的 BMP 圖檔。
4. 針對每張圖片一一設定它們的屬性，決定角色是否可通過、是否為目的地形等等。
5. 最後，存檔成 *FN_TERR_ARCHIVE* 常數所指定的檔名，就可以順利供地圖編輯器及遊戲主程式載入使用。
6. 物品圖庫也依上述步驟如法泡製。

我打算利用同一套程式，相同的介面，幾乎相同的程式碼來進行地形圖庫及物品圖庫的製作及處理。

雙重「物」格的 FTiles

要怎麼做呢？地形圖庫及物品圖庫分別屬於 *TTerrTiles* 及 *TItemTiles* 類別，而不同型態，不同類別的物件通常要分別撰寫程式碼處理才行。別忘了，它們來自同一個父類別－*TTiles*，藉著多型機制的幫忙，我們可以讓 *TTerrTiles* 及 *TItemTiles* 物件共享同一段程式碼，甚至共用同一個變數：

```
#0001  type
#0002    // metaclass, 此類別的 instance 可以是 TTerrTiles or TItemTiles
#0003    TTilesClass = class of TTiles;
#0004
#0005    TMainForm = class(TForm)
#0006    ....
#0007    private
#0008      FTiles: TTiles;
#0009      FFileName: string;
#0010
#0011      procedure ChangeTileClass(TileKind: TTilesClass);
#0012    end;
```

```
#0013
#0014  procedure TMainForm.ChangeTileClass(TileKind: TTilesClass);
#0015  begin
#0016    if Assigned(FTiles) then FTiles.Free;
#0017
#0018    // 依 TileKind 指定的類別來建立 FTiles 物件
#0019    FTiles := TileKind.Create;
#0020  end;
```

你瞧，0008 列宣告著 *TTiles* 類別的 *FTiles* 物件，因爲同一時間只能編輯地形或物品圖庫其中之一，因此我使用同一個 *FTiles* 變數來儲放這兩種物件。

Object Pascal 支援 metaclass 機制，0003 列將 *TTilesClass* 宣告爲 *class of TTiles*，表示 *TTilesClass* 型態變數的值可以是 *TTiles* 類別或是它的衍生類別。我以 *TTilesClass* 型別做爲 *ChangeTileClass* 方法的參數，在此處，*TTilesClass* 型態的 *TileKind* 參數值可能爲 *TTerrTiles* 類別或是 *TItemTiles* 類別。若爲 *TTerrTiles* 類別，則 *FTiles* 變數在執行 0018 行後，會指向一個 *TTerrTiles* 物件，反之亦然。換句話說，我們可以利用 metaclass 機制來表示、傳遞「類別」。

以 RTTI 驗明「物」格

FTiles 指向的不是 *TTerrTiles* 物件就是 *TItemTiles* 物件，那麼要如何辨別它現在所指向的物件究竟是哪一種呢？這要依賴 Object Pascal 的 RTTI（Run-Time Type Information，執行時期型別資訊）機制了，別被這串陌生的英文字嚇到，事實上你可能早已使用 RTTI 而不自覺，那就是其中的 *as* 及 *is* 運算子。這兩個運算子可用來驗證物件的實際類別。例如以下這道運算式：

```
FooObject is TBarClass
```

如果 *FooObject* 屬於 *TBarClass* 類別或它的衍生類別，其值爲真，否則爲假。因此可以使用 *as* 運算子來作保證安全的物件轉型：

```
FooObject as TBarClass
```

這個運算式作用等於下式：

```
TBarClass(FooObject)
```

但使用 *as* 運算子來轉型可以保證不會讓你的程式發生問題。如果這是一個不合法的轉型（*FooObject* 根本不是 *TBarClass* 或其後代的物件），它將會產生 *EInvalidCast* 例外，避免一錯再錯，繼續往下執行。因此下面這道敘述與使用 *as* 運算子來轉型的作用是一模一樣的：

```
if FooObject is TBarClass then
  TBarClass(FooObject)...
```

因爲使用 *as* 及 *is* 運算子比使用直接轉型需要更大的 overhead，因此只要在使用一次 *is* 或 *as* 運算子後，你就可以放心地直接轉型了。所以可以這樣寫：

```
if FooObject is TBarClass then
begin
  TBarClass(FooObject).FieldA := …
  TBarClass(FooObject).FieldB := …
  TBarClass(FooObject).FieldC := …
end;
```

你可以在範例程式中看到我大量地用 *is* 及 *as* 運算子，尤其在事件處理方法內：

```
#0001  procedure TMainForm.mnuSaveClick(Sender: TObject);
#0002  begin
#0003    // 若是"另存新檔" 或還未指定檔名，就先問使用者檔名
#0004    if ((Sender as TComponent).Tag = 1) or (FFileName = '') then
#0005    begin
#0006      dlgSave.Filter := dlgOpen.Filter;
#0007      if not dlgSave.Execute then Exit; //詢問使用者檔名，若按取消就離開
#0008      FFileName := dlgSave.Filename; // 將檔名記起來
#0009    end;
#0010
#0011    FTiles.SaveToFile(FFileName); // 儲存圖庫
#0012    UpdateControlStatus; // 更新視窗標題
#0013  end;
```

VCL 中絕大部分的事件處理方法都帶有一個 *Sender* 參數，代表觸發此方法的物件，因爲不一定是什麼類別，所以 *Sender* 宣告爲所有類別的始祖－*TObject*，但其實它可能是一個 *TButton* 元件、一個 *TImage* 元件甚至一個 *TTimer* 計時器元件。只要我們確定它應該是什麼類別的物件，就可以放心地將它轉型，然後呼叫方法或使用屬性。

以上面的 *mnuSaveClick* 方法爲例，由於我只將這個事件處理方法指派給 *mnuSave* 及 *mnuSaveAs* 兩個 *TMenuItem* 物件，所以當 *mnuSaveClick* 方法被觸發時，理論上 *Sender* 參數必定是兩者其中之一（除非程式有其它地方呼此方法），於是我便可放心地將 *Sender* 參數轉型爲 *TComponent* 類別，取得它的 *Tag* 屬性來使用。爲何轉型爲 *TComponent* 類別，而不轉爲 *TMenuItem* 類別呢？其實兩者都行。只是我個人習慣讓程式碼擁有較大的彈性，若將它轉型爲 *TMenuItem* 類別，萬一日後我又想將 *mnuSaveClick* 方法指派給另一個 *TButton* 物件時，型別轉換部分勢必得修改。因此，我個人的習慣是，若要存取某個屬性或呼叫方法，就將它轉型爲該屬性或方法出現的類別。此處來說，*Tag* 屬性是 *TComponent* 類別介紹的新屬性，因此我就將 *Sender* 參數轉爲 *TComponent* 類別。

提到 *Tag* 屬性，它是所有 VCL 元件都擁有的一個屬性，爲四個位元組的長整數，我們可以任意地使用它。如上述的例子，我將【儲存】及【另儲新檔】兩個元件指派同一個事件處理方法，但代表【儲存】的 *TMenuItem* 元件的 *Tag* 屬性爲 0，而代表【另儲新檔】的 *TMenuItem* 元件的 *Tag* 屬性則設爲 1，以此來區分兩者的不同，進行對應的動作。若我不這樣做，而以傳統的做法分別爲兩個 *TMenuItem* 元件撰寫不同的事件處理方法，就會有很多重覆的程式碼，佔空間，維護起來也較麻煩。

Ouch，離題似乎又遠了~~。於是呢，藉由 *is* 運算子之助，我們可以輕易判別出，目前 *FTiles* 變數指向的究竟是地形或者物品圖庫。

```
if FTiles is TerrTiles then
  …… // 是地形圖庫唷！
else
  …… // 是物品圖庫耶～
```

雙重「物」格變換

什麼情況下，*FTiles* 會指向不同類別的物件呢？只有兩個地方，一是開啓新檔時，詢問使用者準備建立的圖庫類型，二是開啓舊檔時：

```
#0001  procedure TMainForm.mnuNewClick(Sender: TObject);
#0002  begin
#0003    with TTileKindDlg.Create(self) do // 建立詢問對話盒
#0004      try
#0005        if ShowModal = mrOK then // 秀出對話盒,
#0006         // 使用者按下"確定"後會傳回 mrOK
#0007        begin
#0008          if rgptilekind.ItemIndex = 0 then
#0009            ChangeTileClass(TTerrTiles)    // 地形圖庫
#0010          else
#0011            ChangeTileClass(TItemTiles);   // 物品圖庫
#0012
#0013          UpdateControlStatus;
#0014        end;
#0015      finally
#0016        Free; // 無論如何, 摧毀詢問對話盒
#0017      end;
#0018  end;
```

開新檔時比較簡單，我另外設計一個詢問使用者圖庫類型的對話盒，當使用者按下「確定」後會傳回 *mrOK*，再根據它的選擇呼叫 *ChangeTileClass* 方法建立地形或物品圖庫物件。

圖 8-9 / 詢問使用者圖庫類型的對話盒

開啓舊檔就比較有趣了，還記得之前設計 *TTiles* 類別時，在 *TTiles.SaveToFile* 方法中有寫入檔頭標籤及副檔頭標籤嗎？我們就靠著這個及 Object Pascal 的例外捕捉機制（try .. except .. end）來判斷讀入的是何種圖庫，同時建立對應的 *FTiles* 物件：

```
#0001  procedure TMainForm.mnuOpenClick(Sender: TObject);
#0002  begin
#0003    if dlgOpen.Execute then
#0004      try
#0005        try
#0006          ChangeTileClass(TTerrTiles); // 先試試看是否爲地形圖庫
```

```
#0007
#0008         // 讀取有錯的話，就會產生例外，讓我們的例外捕捉
#0009         // 機制處理(except 之後那一段)
#0010         FTiles.LoadFromFile(dlgOpen.Filename);
#0011         FFileName := dlgOpen.Filename; // 讀取地形圖庫成功
#0012       except
#0013         // 如果不是地形圖庫，檢查副檔頭標籤時會產生例外，
#0014         // 所以跳到這兒來執行
#0015
#0016         // 再試試看是否爲物品圖庫，否則就什麼都不是
#0017         ChangeTileClass(TItemTiles);
#0018
#0019         // 有錯的話，也會產生例外，我們就不處理了
#0020         FTiles.LoadFromFile(dlgOpen.Filename);
#0021         FFileName := dlgOpen.Filename;
#0022       end;
#0023     finally
#0024       UpdateControlStatus;
#0025     end;
#0026  end;
```

讀取圖庫時採取「**試誤法**」，先試試看是否爲地形圖庫，不是的話，再試試看是否爲物品圖庫，否則就不理它。我憑藉的是 *TTiles.LoadFromFile* 方法中會呼叫 *CheckSignature* 函式來檢查檔頭標籤，而 *CheckSignature* 函式會在檔頭標籤與預期不符時丟出一個例外：

```
#0001  procedure CheckSignature(fs: TFileStream; const Sig: string);
#0002  var
#0003    S: string;
#0004  begin
#0005    SetLength(S, Length(Sig));
#0006    fs.Read(S[1], Length(Sig)); // 從 TFileStream 讀出檔頭標籤
#0007    if CompareText(S, Sig) <> 0 then // 丟出例外
#0008      raise Exception.Create('File signature not match');
#0009  end;
```

這個看起來很 dirty 又很 smart 的「**試誤法**」，其實是懶得另外撰寫檢查檔頭標籤函式的結果，不然正常的寫法會像是這樣，你喜歡哪個呢？

```
if IsTerrArchive(dlgOpen.Filename) then
begin
  ChangeTileClass(TTerrTiles);
  FTiles.LoadFromFile(dlgOpen.Filename);
end else begin
  ChangeTileClass(TItemTiles);
  FTiles.LoadFromFile(dlgOpen.Filename);
```

```
end;
```

繪製圖庫圖片

視窗中央那個 *TDrawGrid* 物件 *grdPreview* 會根據目前的 *FTiles* 內容將圖片顯示出來，*TDrawGrid* 元件本身並不儲存任何資訊，顯示的結果端視我們如何處理它的 *OnDrawCell* 事件而定。以下是 *grdPreview* 的 *OnDrawCell* 事件處理方法：

```
#0001  procedure TMainForm.grdPreviewDrawCell(Sender: TObject; ACol,
#0002      ARow: Integer; Rect: TRect; State: TGridDrawState);
#0003  var
#0004    No: Integer;
#0005  begin
#0006    No := ARow * grdPreview.ColCount + ACol; // 由行及列換算圖片編號
#0007
#0008    with grdPreview.Canvas do
#0009      if No < FTiles.TileNum then // 將對應的圖形畫出來
#0010        BitBlt(Handle, Rect.Left, Rect.Top, TILE_WIDTH, TILE_HEIGHT,
#0011          FTiles.Bitmap.Canvas.Handle, FTiles.TilePos_Left[No],
#0012          FTiles.TilePos_Top[No], SRCCOPY)
#0013      else
#0014      begin
#0015        Brush.Color := clBlack; // 編號大於圖片數目, 塗滿黑色
#0016        FillRect(Rect);
#0017      end;
#0018  end;
```

同樣地，還是經由 *TTiles* 的 *TilePos_Left* 及 *TilePos_Top* 兩個屬性來取得圖片在圖庫中的座標，再利用 *BitBlt* API 將該圖片貼到 *grdPreview* 的對應位置上；對於沒有圖片的那些圖格，就塗黑。

而每當使用者選擇 *grdPreview* 上的某一格時，*grdPreview* 的 *OnSelectCell* 事件處理方法 *grdPreviewSelectCell* 就必須先將目前設定的屬性寫回 *FTiles* 物件中，接著再讀出即將選擇的圖片屬性，依屬性更新 *TCheckBox* 元件狀態：

```
#0001  procedure TMainForm.grdPreviewSelectCell(Sender: TObject; ACol,
#0002      ARow: Integer; var CanSelect: Boolean);
#0003  var
#0004    OldNo, No: Integer;
#0005  begin
```

```
#0006   with grdPreview do
#0007   begin
#0008     //  計算原本選擇的圖片編號
#0009     OldNo := Row * grdPreview.ColCount + Col;
#0010     if OldNo < FTiles.TileNum then // 將使用者設定的圖片屬性寫回去
#0011     begin
#0012       if FTiles is TTerrTiles then // 若是地形圖庫的話...
#0013       begin
#0014         // 根據 cbxCanPass 及 cbxTarget 兩個 checkbox 來設定圖片屬性
#0015         TTerrTiles(FTiles).Attrs[OldNo] := [];
#0016
#0017         // 可以穿越的地形
#0018         if cbxCanPass.Checked then TTerrTiles(FTiles).Attrs[OldNo]
#0019           := TTerrTiles(FTiles).Attrs[OldNo] + [taCanPass];
#0020
#0021         // 它是目的地形
#0022         if cbxTarget.Checked then TTerrTiles(FTiles).Attrs[OldNo] :=
#0023           TTerrTiles(FTiles).Attrs[OldNo] + [taTarget];
#0024       end else
#0025       begin // 若是物品圖庫的話...
#0026         TItemTiles(FTiles).Attrs[OldNo] := [];
#0027
#0028         // 可以搬動的物品
#0029         if cbxCanMove.Checked then TItemTiles(FTiles).Attrs[OldNo]
#0030           := TItemTiles(FTiles).Attrs[OldNo] + [iaCanMove];
#0031
#0032         // 覆蓋用物品
#0033         if cbxSource.Checked then TItemTiles(FTiles).Attrs[OldNo] :=
#0034           TItemTiles(FTiles).Attrs[OldNo] + [iaSource];
#0035       end;
#0036     end;
#0037   end;
#0038
#0039   No := ARow * grdPreview.ColCount + ACol; // 計算即將選擇的圖片編號
#0040   if No < FTiles.TileNum then // 秀出目前所選的圖片屬性
#0041   begin
#0042     lblTileNo.Caption := Format('編號: %d',[No]);
#0043     // 將所選擇的圖片屬性由 checkbox 元件表現出來
#0044     if FTiles is TTerrTiles then
#0045     begin
#0046       cbxCanPass.Checked := taCanPass in
#0047         TTerrTiles(FTiles).Attrs[No];
#0048       cbxTarget.Checked := taTarget in TTerrTiles(FTiles).Attrs[No];
#0049     end else
#0050     begin
#0051       cbxCanMove.Checked := iaCanMove in
```

```
#0052         TItemTiles(FTiles).Attrs[No];
#0053        cbxSource.Checked := iaSource in TItemTiles(FTiles).Attrs[No];
#0054      end;
#0055    end else CanSelect := False; // 選擇不合法的圖格, 不給選
#0056  end;
```

嗯，到此為止，也許你不相信，但是圖庫編輯器一不小心就這樣完工了，圖 8-10 及圖 8-11 分別是圖庫編輯器在製作地形及物品圖庫時的執行畫面。從畫面中可以看到，地形我只提供四張圖片，編號 0 號為草地，以它做為預設地形；物品圖片更少，只有一顆足球，放在編號 1 的位置上，因為編號 0 具有特殊意義，代表「此地無任何物品」。

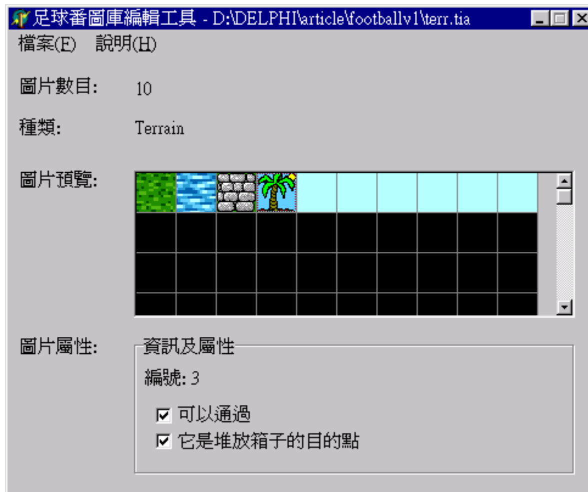

圖 8-10 / 圖庫編輯器的執行畫面（地形圖庫）

圖 8-11 / 圖庫編輯器的執行畫面（物品圖庫）

事實上這大概是全世界最陽春的圖庫編輯器了，跟圖 8-2 及圖 8-3 列出的 StarCraft 及英雄無敵 III 的地圖編輯器一比，咱們的圖庫編輯器羞得無地自容，差點離家出走，還是我好說歹說才將它留住。我想，就算是第二陽春的圖庫編輯器至少也有圖片拉曳、更換位置編號等功能，再者圖片群組及物件的概念也該支援，可以發揮的地方還多得是，讓我們以後慢慢玩吧。

地圖編輯器

有了圖庫編輯器，製作出圖庫後，接著就可以編輯地圖。地圖編輯器的目的很簡單－提供一個 WYSIWYG 的介面讓使用者可以方便地編輯存放那兩層地圖的二維陣列元素值。呵，很繞口吧。

說得清楚點，*TMap* 不是擁有兩個 *TMapArray* 型態的 *TerrMap* 及 *ItemMap* 變數嗎？*TMapArray* 型態是 *TILE_NUM_X* * *TILE_NUM_Y* 大小的二維 *Byte* 陣列，而**地圖編輯器的目的就是提供與遊戲進行時相同的圖片及畫面以及方便的操作介面，供關卡設計者編輯陣列內容。**

目的很簡單，說來只有幾句話，但寫來比我們那個世界第一笨的圖庫編輯器還稍微複雜一滴滴（以程式碼行數比較的話:p）。還是先將介面設計出來：

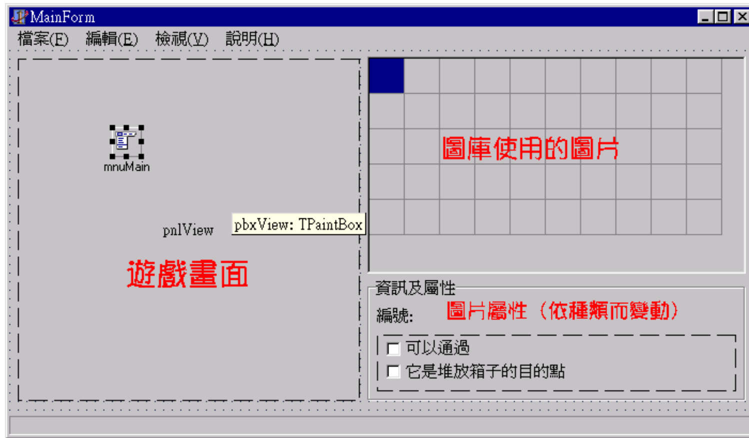

圖 8-12 / 地圖編輯器的設計畫面

程式規劃如下：

1. 有三種編輯模式，分別是地形，物品及角色編輯模式。
2. 與遊戲畫面不同的是，可以顯示格線及特殊區域以利編輯。
3. 能夠檢查關卡地形及物品擺設是否合法。

首先宣告編輯模式型別及視窗類別 *TMainForm*：

```
#0001  type
#0002    // 三種編輯模式: 地形, 物品及角色
#0003    TEditKind = (ekTerrs, ekItems, ekRole);
#0004
#0005    TMainForm = class(TForm)
#0006    ...
#0007    private
#0008      FEditKind: TEditKind; // 目前編輯模式
#0009      FLevelNo: Integer; // 目前編輯的關卡
#0010      FModified: Boolean; // 載入關卡後是否更動過
#0011
#0012      FRole: TRole; // 角色物件
#0013      FBackBitmap: TBitmap; // double-buffering 用的背景 bitmap
#0014
```

```
#0015     FCursorX, FCursorY: Integer; // 滑鼠游標位置
#0016     FButtonPressed: TMouseButton; // 滑鼠按鍵狀態
#0017
#0018     procedure DrawBackBitmap; // 繪製背景 bitmap
#0019     procedure UpdateView; // 更新編輯畫面
#0020     procedure UpdateControlStatus;
#0021
#0022     procedure SetLevelNo(Value: Integer);
#0023
#0024     procedure MySaveFile; // 儲存地圖檔案
#0025     function AskSaveMap: Boolean; // 確認是否儲存地圖
#0026     function ValidateMap: Boolean; // 檢查地圖是否合法
#0027
#0028     property LevelNo: Integer read FLevelNo write SetLevelNo;
#0029   public
#0030   end;
```

0012 列宣告了 *TRole* 物件 *FRole*，這是 *TRole* 類別第一回派上用場呢。不過在此只是虛晃幾招，只用它來做顯示及定位角色的用途而已，那複雜的移動邏輯仍派不上用場。

0013 列的 *FBackBitmap* 就是文章前頭談到 double-buffering 時所說的背景 bitmap。在 *UpdateView* 方法中，會先呼叫 *DrawBackBitmap* 將應該出現在畫面上的所有東東繪製於 *FBackBitmap*，再呼叫 *TCanvas.Draw* 方法一口氣將 *FBackBitmap* 畫上 *pbxView*，也就是代表編輯畫面的 *TPaintBox* 畫布上頭。所以 *UpdateView* 方法只有簡單的短短兩行：

```
#0001  procedure TMainForm.UpdateView;
#0002  begin
#0003    DrawBackBitmap; // 將畫面放到 FBackBitmap
#0004    pbxView.Canvas.Draw(0, 0, FBackBitmap); // 複製到 pbxView
#0005  end;
```

在地圖編輯器中，因為必須同時使用地形圖庫及物品圖庫，因此就不像圖庫編輯器那樣，兩者共用一個 *TTiles* 變數，而直接使用宣告在 Tiles 單元中的全域變數 *Terrs* 及 *Items*。

程式初始化

在 *OnCreate* 事件處理方法中，初始化所有的物件，並載入角色圖片及地形、物品圖庫，另外還設定好 *FBackBitmap*，讓它跟編輯畫面一樣大：

```
#0001  procedure TMainForm.FormCreate(Sender: TObject);
#0002  begin
#0003    FEditKind := ekTerrs; // 預設爲地形編輯模式
#0004    FButtonPressed := mbMiddle; // 表示目前滑鼠鍵沒有按著
#0005
#0006    FRole := TRole.Create; // 主角物件
#0007
#0008    Terrs := TTerrTiles.Create; // 地形圖片及資訊
#0009    Items := TItemTiles.Create; // 物品圖片及資訊
#0010
#0011    Map := TMap.Create; // 關卡地圖
#0012    try
#0013      FRole.LoadBits; // 讀入主角的圖形
#0014
#0015      Terrs.LoadFromFile(AppDir + FN_TERR_ARCHIVE); // 讀入地形
#0016      Items.LoadFromFile(AppDir + FN_ITEM_ARCHIVE); // 讀入物品
#0017    except
#0018      on E: Exception do // 一旦有錯，就顯示錯誤訊息，然後關閉程式
#0019        begin
#0020          ShowMessage(E.message);
#0021          // 以 asynchronous 方式關閉視窗
#0022          PostMessage(Handle, WM_CLOSE, 0, 0);
#0023        end;
#0024    end;
#0025
#0026    // According to tile num and size, adjust dimension of pnlView
#0027    pnlView.ClientWidth := TILE_WIDTH * TILE_NUM_X;
#0028    pnlView.ClientHeight := TILE_HEIGHT * TILE_NUM_Y;
#0029
#0030    FBackBitmap := TBitmap.Create;   // bitmap for double-buffering
#0031    FBackBitmap.Width := TILE_WIDTH * TILE_NUM_X; //大小與編輯畫面一致
#0032    FBackBitmap.Height := TILE_HEIGHT * TILE_NUM_Y;
#0033
#0034    LevelNo := 1; // 預設爲第一關
#0035    UpdateControlStatus;
#0036
#0037    // 滑鼠游標位置
#0038    FCursorX := -1;
#0039    FCursorY := -1;
#0040  end;
```

0015 及 0016 列載入圖庫檔案時所用的 *AppDir* 字串記錄著程式執行檔所在的目錄，由程式庫的 xFiles 單元提供。

0030 ~ 0032 列建立 double-buffering 用的背景 bitmap *FBackBitmap*，並將大小設定與遊戲

畫面相同，這使得 *UpdateView* 方法中，將 *FBackBitmap* 內容複製到編輯畫面 *pbxView* 的動作省事許多。

奇妙的 LevelNo 屬性

LevelNo 是整數型態的屬性，每當設定新值時，就會呼叫 *SetLevelNo* 方法去設定（見類別宣告 0028 列）：

```
#0001  procedure TMainForm.SetLevelNo(Value: Integer);
#0002  begin
#0003    try
#0004      // 讀取地圖檔及角色位置
#0005      Map.LevelNo := Value;
#0006    except
#0007      // 若讀取地圖檔失敗, 清除整張地圖
#0008      Map.ResetMap;
#0009    end;
#0010
#0011    FRole.X := Map.Role_X; // 將角色位置由 Map 物件中抄出來
#0012    FRole.Y := Map.Role_Y;
#0013
#0014    FLevelNo := Value;
#0015    FModified := False; // 地圖尚未更動（當然:p）
#0016    UpdateControlStatus;
#0017    UpdateView; // 別忘了更新編輯畫面
#0018  end;
```

這就是我爲什麼喜歡使用屬性的原因。瞧 *FormCreate* 方法中簡簡單單的一道敘述，將 *LevelNo* 設爲 1，事實上它的屬性寫入方法－*SetLevelNo* 就會爲我讀取第一關的地圖出來，同時取得角色位置，設定其它變數，並且更新編輯畫面等等，多麼優雅的撰寫方式呀。屬性機制讓所有讀／寫的邊際效應都漂亮地隱藏在屬性值讀／取的簡單敘述背後。

LevelNo 屬性甚至可以這樣使用：

```
#0001  procedure TMainForm.mnuRestoreLevelClick(Sender: TObject);
#0002  begin
#0003    LevelNo := LevelNo;
#0004  end;
```

mnuRestoreLevelClick 是選擇【恢復此關卡原狀】選項的事件處理方法，看到上面那行程式碼，不曉得 *LevelNo* 是屬性的傢伙一定會認為我花轟了，竟然把一個變數指派給自己，只是浪費 CPU 時間的無意義動作呀。但是別忽略隱藏在 *LevelNo* 屬性背後的 *SetLevelNo* 方法，將 *LevelNo* 指派給 *LevelNo* 的結果是，*SetLevelNo* 方法會去重新載入目前的關卡地圖，達成「恢復此關卡原狀」的目的，很特別吧。

繪製編輯畫面

前頭剛提過，呈現編輯畫面的核心方法只有一個－*DrawBackBitmap*，它先繪製地形層，接著物品層，最後是角色。可以想像出，待會才要進行的遊戲主程式的 *DrawBackBitmap* 方法，似乎也只需要這三個動作就夠了。但在地圖編輯器中，還要能夠顯示格線及特殊區域，所以繪製物品層後，繪製角色前，另外加上三段程式碼，分別將格線，目的地形及覆蓋用物品標示出來：

```
#0001  procedure TMainForm.DrawBackBitmap;
#0002  var
#0003    X, Y: Integer;
#0004  begin
#0005    Map.DrawTerrMap(FBackBitmap.Canvas); // 繪製地形層
#0006    Map.DrawItemMap(FBackBitmap.Canvas); // 繪製物品層
#0007
#0008    // 如果使用者想看格線, 就將格線畫出來
#0009    if mnuShowGrid.Checked then
#0010      with FBackBitmap.Canvas do
#0011      begin
#0012        Pen.Color := clBlack; // 黑色
#0013        Pen.Style := psSolid; // 實線
#0014        Pen.Width := 1; // 寬度為 1
#0015
#0016        // 先畫橫線
#0017        for Y := 0 to TILE_NUM_Y - 1 do
#0018        begin
#0019          MoveTo(0, Y * TILE_HEIGHT);
#0020          LineTo(TILE_NUM_X * TILE_WIDTH, Y * TILE_HEIGHT);
#0021        end;
#0022
#0023        // 再畫直線
#0024        for X := 0 to TILE_NUM_X - 1 do
```

```
#0025       begin
#0026         MoveTo(X * TILE_WIDTH, 0);
#0027         LineTo(X * TILE_WIDTH, TILE_NUM_Y * TILE_HEIGHT);
#0028       end;
#0029     end;
#0030
#0031   // 如果使用者想看特殊區域，將目的地地形標出來
#0032   if mnuShowSpeicalArea.Checked then
#0033     with FBackBitmap.Canvas do
#0034     begin
#0035       Pen.Color := clRed; // 紅色
#0036       Pen.Width := 1; // 寬度爲 1
#0037       Pen.Style := psDash; // 虛線
#0038       Brush.Style := bsClear;
#0039
#0040       for Y := 0 to TILE_NUM_Y - 1 do // 逐一檢查
#0041         for X := 0 to TILE_NUM_X - 1 do
#0042           if Map.IsTarget[X, Y] then
#0043           begin // 外框加上右上畫到左下的紅色虛線
#0044             Rectangle(X * TILE_WIDTH, Y * TILE_HEIGHT, (X + 1) *
#0045               TILE_WIDTH - 1, (Y + 1) * TILE_HEIGHT - 1);
#0046             MoveTo((X + 1) * TILE_WIDTH, Y * TILE_HEIGHT);
#0047             LineTo(X * TILE_WIDTH, (Y + 1) * TILE_HEIGHT);
#0048           end;
#0049     end;
#0050
#0051   // 如果使用者想看特殊物品，將覆蓋用物品標出來
#0052   if mnuShowSpeicalItems.Checked then
#0053     with FBackBitmap.Canvas do
#0054     begin
#0055       Pen.Color := clLime; // 亮綠色
#0056       Pen.Width := 1; // 寬度爲 1
#0057       Pen.Style := psDash; // 虛線
#0058       Brush.Style := bsClear;
#0059
#0060       for Y := 0 to TILE_NUM_Y - 1 do
#0061         for X := 0 to TILE_NUM_X - 1 do
#0062           if Map.IsSource[X, Y] then
#0063           begin // 外框加上左上畫到右下的亮綠色虛線
#0064             Rectangle(X * TILE_WIDTH, Y * TILE_HEIGHT, (X + 1) *
#0065               TILE_WIDTH - 1, (Y + 1) * TILE_HEIGHT - 1);
#0066             MoveTo(X * TILE_WIDTH, Y * TILE_HEIGHT);
#0067             LineTo((X + 1) * TILE_WIDTH, (Y + 1) * TILE_HEIGHT);
#0068           end;
#0069     end;
#0070
```

```
#0071    FRole.Draw(FBackBitmap.Canvas); // 最後畫出角色圖案
#0072  end;
```

你可以先偷偷看一下圖 8-13，看看這段程式碼繪出的畫面長得什麼樣子。當然囉，可以獨立開關這三個顯示選項，預設值爲關，所以你自己執行時不會看到格線及特殊區域，選取功能表將它們打開後才看得到。

圖 8-13 / 地圖編輯器的執行畫面（將三個顯示選項都打開了）

右上方顯示圖庫圖片的 *grdView* 及右下角顯示圖片屬性的 *TCheckBox* 都跟圖庫編輯器中一樣，因此不再贅述。

選取【編輯 /（地形、物品、人物）】時，會觸發 *mnuEditRoleClick* 方法，首先將觸發此方法的 *TMenuItem* 元件打勾（即 *Checked* 屬性設爲 *True*），設定 *FEditKind* 的新值，此處我又使用前述的方法，將三個 *TMenuItem* 元件的 *OnClick* 事件全指派到同一個事件處理方法，以 *Tag* 屬性區分彼此。所以 0009 列只要輕鬆的一行指派敘述，到 *Kinds* 陣列中查表， 就可以將 *FEditKind* 設定爲對應的編輯模式。此方法的其它部分都在處理切換編輯模式時控制項的介面差異問題（顯示顏色，控制項是否致能等等）。

```
#0001  procedure TMainForm.mnuEditRoleClick(Sender: TObject);
#0002  const
#0003    Kinds: array[0..2] of TEditKind  = (ekTerrs, ekItems, ekRole);
#0004    Colors: array[Boolean] of TColor = (clBtnFace, clWindow);
#0005  var
#0006    bEnable: Boolean;
```

```
#0007  begin
#0008    (Sender as TMenuItem).Checked := True;
#0009    // 設定選取的編輯模式
#0010    FEditKind := Kinds[(Sender as TMenuItem).Tag];
#0011
#0012    UpdateControlStatus;
#0013    UpdateView; // 更新畫面
#0014
#0015    ...
#0016
#0017  end;
```

滑鼠的拉曳及物品置放

真正的好菜現在上桌，不論在何種編輯模式下，當滑鼠游標在編輯畫面範圍內時，游標底下對映的圖格就會有個水藍色的框框跟著它跑，這是怎麼做到的呢？這效果來自於 *grdView* 的 *OnMouseMove* 事件處理方法 *grdViewMouseMove*：

```
#0001  procedure TMainForm.pbxViewMouseMove(Sender: TObject; Shift:
#0002    TShiftState; X, Y: Integer);
#0003  begin
#0004    // 已經跑出範圍外了...
#0005    if (X < 0) or (X >= TILE_WIDTH * TILE_NUM_X) or (Y < 0) or
#0006      (Y >= TILE_HEIGHT * TILE_NUM_Y) then Exit;
#0007
#0008    FCursorX := X div TILE_WIDTH; // 換算座標，以圖格爲單位
#0009    FCursorY := Y div TILE_HEIGHT;
#0010
#0011    // 一邊拉曳一邊置放物品的效果
#0012    pbxViewMouseDown(Sender, FButtonPressed, Shift, X, Y);
#0013
#0014    UpdateView; // 重繪編輯畫面
#0015
#0016    // 畫出目前所選取的區域外框
#0017    with pbxView.Canvas do
#0018    begin
#0019      Pen.Width := 2;
#0020      Pen.Color := clBlue;
#0021      Brush.Style := bsClear;
#0022      Rectangle(FCursorX * TILE_WIDTH, FCursorY * TILE_HEIGHT,
#0023        (FCursorX + 1) * TILE_WIDTH, (FCursorY + 1) * TILE_HEIGHT);
#0024    end;
#0025
```

```
#0026   stbMain.SimpleText := Format('Position: (%d, %d)',[FCursorX,
#0027     FCursorY]);
#0028  end;
```

每當滑鼠指標在控制項上移動時，就會不斷產生 *OnMouseMove* 事件。那麼，你也許會問，既然如此，那麼滑鼠指標一定是在控制項範圍內呀，又何必要有 0005 ~ 0006 列的範圍檢查碼呢？原因是，若使用者在控制項內按下滑鼠任一鍵然後「拉曳」的話，此控制項就會不斷地收到 *OnMouseMove* 事件，不論滑鼠指標是否早已移出控制項範圍，直到滑鼠鍵放開，所以 0005 ~ 0006 列的範圍檢查是必要的。

0008 ~ 0009 列將座標值由以像素爲單位換算爲以圖格爲單位。0012 列是爲了達成一邊拉曳滑鼠一邊置放物品的效果，主動觸發 *pbxView* 的 *OnMouseDown* 事件處理方法以設定或清除圖格。接著，畫出所選取區域的外框。藍色外框是仿英雄無敵 III 地圖編輯器而來的，我覺得挺顯眼好看。

接著介紹最重要的一個，也就是達成地圖編輯器目的的重要方法－修改地圖內容的 *pbxView OnMouseDown* 事件處理方法：

```
#0001  procedure TMainForm.pbxViewMouseDown(Sender: TObject; Button:
#0002    TMouseButton; Shift: TShiftState; X, Y: Integer);
#0003  var
#0004    No: Integer;
#0005  begin
#0006    // 將按下的按鍵記錄起來, 配合 OnMouseMove event handler
#0007    // 產生拉曳設定效果
#0008    FButtonPressed := Button;
#0009
#0010    if Button = mbMiddle then Exit; // 滑鼠中鍵不做任何事
#0011
#0012    with grdPreview do
#0013      No := Row * ColCount + Col; // 計算目前選擇的圖片編號
#0014
#0015    if Button = mbLeft then // 左鍵是設定
#0016    begin
#0017      case FEditKind of // 根據編輯模式不同進行不同的設定動作
#0018        ekTerrs:
#0019          Map.TerrMap[FCursorX, FCursorY] := No; // 將新地形擺上
#0020
#0021        ekItems:
#0022        begin
```

```
#0023        // 物品不可擺在角色身上
#0024        if (FRole.X = FCursorX) and (FRole.Y = FCursorY) then Exit;
#0025
#0026        Map.ItemMap[FCursorX, FCursorY] := No; // 將新物件擺上
#0027      end;
#0028
#0029      ekRole:
#0030      begin
#0031        // 角色不可以擺在物品上
#0032        if Map.ItemMap[FCursorX, FCursorY] <> 0 then Exit;
#0033        // 角色不可以擺在不可走動的地形上
#0034        if not Map.CanPass[FCursorX, FCursorY] then Exit;
#0035
#0036        FRole.X := FCursorX; // 設定角色位置
#0037        FRole.Y := FCursorY;
#0038      end;
#0039    end;
#0040  end else // 右鍵是清除
#0041  begin
#0042    case FEditKind of
#0043      ekTerrs: Map.TerrMap[FCursorX, FCursorY] := 0; // 清除地形
#0044
#0045      ekItems: Map.ItemMap[FCursorX, FCursorY] := 0; // 清除物品
#0046
#0047      ekRole: ; // 角色不能清掉, so do nothing
#0048    end;
#0049  end;
#0050
#0051  FModified := True;
#0052  UpdateView;
#0053 end;
```

這段程式碼的註解相當清楚，邏輯也十分簡單，首先判斷使用者按下的是左鍵或右鍵，左鍵代表置放，右鍵代表清除。接著再根據目前的編輯模式，置放地形、物品、角色或是清除地形或物品。

唯一要注意的就是置放物品及角色前，要小心會不會讓地圖產生不合法，或是遊戲無法進行的窘況，例如角色不可擺在物品上，也不可擺在無法穿越的地形上這類的合法性檢查。

FButtonPressed 變數是用來實作拉曳設定效果的關鍵處，即是你可以按著滑鼠左鍵隨意在遊戲畫面上游走，經過之處就會擺上目前選擇的地形或物品，編輯起來爽快多了。

FButtonPressed 是 *TMouseButton* 列舉型態，其值可能為 *mbLeft*、*mbRight*、*mbMiddle* 三者之一，分別代表滑鼠左鍵、右鍵及中鍵。在此利用 *FButtonPressed* 記錄著使用者目前按下的滑鼠鍵，原本還須用一個布林變數來記錄目前是否真正按著滑鼠鍵，但因為我們沒用到中鍵，所以設定當 *FButtonPressed = mbMiddle* 時，就代表滑鼠鍵沒有按著，其值為 *mbLeft* 或 *mbRight* 時，才代表滑鼠左鍵或右鍵正被按壓著。

擁有 *FButtonPressed* 資訊後，就可以在 *OnMouseMove* 事件處理方法中，主動呼叫 *OnMouseDown* 的事件處理方法，設定或清除對應的圖格。哦對了，別忘了撰寫 *OnMouseUp* 事件處理方法，在滑鼠鍵放開時，將 *FButtonPressed* 設為 *mbMiddle*：

```
procedure TMainForm.pbxViewMouseUp(Sender: TObject; Button: TMouseButton;
    Shift: TShiftState; X, Y: Integer);
begin
  FButtonPressed := mbMiddle; // 設定為 mbMiddle 表示使用者已放開滑鼠鍵
end;
```

關卡合法性檢查

地圖編輯器還有一項重要的功能，就是在關卡設計完成後，儲存入檔案前，檢查地圖的合法性，覆蓋用物品及目的地形數目，角色是否位於不能移動的地形上等等，合法的關卡才能順利地進行遊戲。檢查地圖合法性的 *ValidateMap* 方法程式碼如下：

```
#0001  // 檢查地圖是否合法
#0002  function TMainForm.ValidateMap: Boolean;
#0003  var
#0004    X, Y                   : Integer;
#0005    SourceCount, TargetCount: Integer;
#0006  begin
#0007    Result := False;
#0008
#0009    SourceCount := 0; // 覆蓋用物品數目
#0010    TargetCount := 0; // 目的地形數目
#0011    for Y := 0 to TILE_NUM_Y - 1 do
#0012      for X := 0 to TILE_NUM_X - 1 do
#0013      begin
#0014        // 計算覆蓋用物品及目的地形數目
#0015        if Map.IsSource[X, Y] then Inc(SourceCount);
#0016        if Map.IsTarget[X, Y] then Inc(TargetCount);
```

```
#0017
#0018        // 此格地圖中記錄的圖片編號是不是大於圖庫的圖片數目 ?
#0019          是的話就調回預設值
#0020        if Map.TerrMap[X, Y] > Terrs.TileNum then
#0021          Map.TerrMap[X, Y] := 0;
#0022
#0023        if Map.ItemMap[X, Y] > Items.TileNum then
#0024          Map.ItemMap[X, Y] := 0;
#0025      end;
#0026
#0027    // 是否沒有目的地形 ? (無法過關)
#0028    if (TargetCount = 0) and not YesNoBox('沒有目的地形, 是否繼續?')
#0029      then Exit;
#0030
#0031    // 是否覆蓋用物品少於目的地形 ? (無法過關)
#0032    if (TargetCount > SourceCount) and not
#0033      YesNoBox('覆蓋用物品少於目的地形, 是否繼續?') then Exit;
#0034
#0035    // 角色位於不能移動的地形上
#0036    if not Map.CanPass[Map.Role_X, Map.Role_Y] and not
#0037      YesNoBox('主角位於不能移動的地形上, 是否繼續?') then Exit;
#0038
#0039    Result := True; // 此關卡通過檢查
#0040  end;
```

至此，地圖編輯器也順利完工，讓我們再看一次執行畫面，這回三個特殊顯示選項沒有打開。嗯，遊戲畫面也可從這兒的編輯畫面看出大概了，是不是對即將完成的遊戲更充滿期待呢？

圖 8-14 / 地圖編輯器的執行畫面

明明是倉庫番類型的遊戲，不過物品圖庫竟然沒有箱子的蹤影，嘻，這是因為我找不到箱子的圖片，只好以足球代替，這也是這套遊戲之所以稱為「足球番」的原因。:p 無論如何，這套「足球番」已呼之欲出，加把勁就要完成了，休息一下，咱們繼續。

「足球番」主程式

一路過關斬將，砍了圖庫編輯器，宰掉地圖編輯器，最後來到大魔王－「足球番」主程式前...

老法子，先在將視窗介面設計好。在設計時期看起來，這遊戲主程式比前兩支程式都還簡單，因為只有一個 *TPaintBox* 元件 *pbxView*，上面再放著一個 *TMainMenu* 元件及三個計時器元件而已。

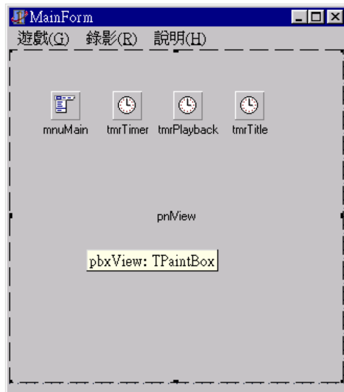

圖 8-15 / 「足球番」主程式的設計畫面

0003 列首先宣告 *TGameStatus* 型態，定義四種狀態，分別是「歡迎畫面」、「遊戲進行中」、「關卡完成後的慶祝畫面」及「過關動作回顧」（即重播功能）。有許多變數，物件及方法的命名及含意都與地圖編輯器一模一樣，如代表角色的 *FRole*，擔任 double-buffering 緩衝區的 *FBackBitmap*，繪製背景 bitmap 的 *DrawBackBitmap* 方法及更新遊戲畫面的 *UpdateView* 方法等等。比較新鮮的是存放目前遊戲狀態的 *FGameStatus* 變

數，記錄可以播放過關回顧關卡編號的 *FPlayBackLevelNo* 變數，可在遊戲畫面中央或正上方畫出字串及方框的 *DrawStatusBox* 方法，以及檢查是否已完成任務的 *CheckFinished* 方法等。類別宣告的原始程式碼列表如下：

```
#0001  type
#0002    // 有四種狀態, Title 畫面, 遊戲中, 完成某關卡及過關回顧
#0003    TGameStatus = (gsTitle, gsPlaying, gsSuccess, gsPlayback);
#0004
#0005    TMainForm = class(TForm)
#0006    ...
#0007    private
#0008      FGameStatus: TGameStatus; // 遊戲狀態
#0009      FLevelNo, FPlayingTime: Integer; // 目前關卡及遊戲進行時間
#0010      FRole: TRole; // 角色物件
#0011
#0012      FBackBitmap: TBitmap; // 用來做 double-buffering 的 bitmap
#0013      FPlayBackLevelNo: Integer; // 可以 playback 的 LevelNo
#0014
#0015      procedure DrawBackBitmap; // 繪製背景 bitmap
#0016      procedure UpdateView; // 更新遊戲畫面
#0017      procedure UpdateControlStatus;
#0018
#0019      // 在遊戲畫面中央或正上面繪出字串及外圍方框
#0020      procedure DrawStatusBox(S: string; TopOrCenter: Boolean);
#0021      function CheckFinished: Boolean; // 檢查是否已完成任務
#0022
#0023      procedure SetGameStatus(Value: TGameStatus);
#0024      procedure SetLevelNo(Value: Integer);
#0025
#0026      property GameStatus: TGameStatus read FGameStatus write
#0027        SetGameStatus;
#0028      property LevelNo: Integer read FLevelNo write SetLevelNo;
#0029    public
#0030    end;
```

TMainForm 的 *OnCreate* 事件處理方法與地圖編輯器中幾乎完全一樣：同樣地建立及初始化 *TRole*、*TMap*、*TTiles* 物件及緩衝用 bitmap 等等，並沒有新的工作。

三個小時鐘

而丟在 form 上的三個 *TTimer* 計時器元件，它們的任務是什麼呢？

- *tmrTitle*

  負責在歡迎畫面時，讓角色快速地亂數移動，產生爆笑效果，觸發間隔設為 50 毫秒。

- *tmrTimer*

  為遊戲進行中的計時器，每 1000 毫秒，即一秒鐘觸發一次，同時更新螢幕上的時鐘，讓使用者曉得此關卡已花費多少時間。

- *tmrPlayback*

  重播過關記錄時，每 500 毫秒觸發一次，讓角色每半秒鐘根據先前的動作記錄移動到下一步，若不使用計時器來放慢速度，直接用迴圈來播放，幾百步驟的動作記錄可能一眨眼就播完了:p

它們的觸發事件處理方法分別如下：

```
#0001  procedure TMainForm.tmrTimerTimer(Sender: TObject);
#0002  begin
#0003    Inc(FPlayingTime); // 遊戲進行時間加一秒
#0004    UpdateView; // 強制更新畫面
#0005  end;
#0006
#0007  procedure TMainForm.tmrTitleTimer(Sender: TObject);
#0008  begin
#0009    FRole.Move(TDirection(Random(4))); // 讓主角任意走動
#0010    UpdateView; // 更新畫面
#0011  end;
#0012
#0013  procedure TMainForm.tmrPlaybackTimer(Sender: TObject);
#0014  begin
#0015    // 按照之前的動作循序走動
#0016    // 使用 tmrPlayback 的 Tag 屬性來記錄目前走到第幾步
#0017    FRole.Move(TDirection(FRole.PlaybackList[tmrPlayback.Tag]));
#0018    // 這一步走過了, 遞增至下一步
#0019    tmrPlayback.Tag := tmrPlayback.Tag + 1;
#0020
#0021    // 全部播完了, 進入過關畫面
#0022    if tmrPlayback.Tag = FRole.Playbacklist.Count then
#0023      GameStatus := gsSuccess;
#0024
#0025    UpdateView; // 更新畫面
#0026  end;
#0027
#0028  initialization
```

```
#0029   Randomize; // 重播亂數種子
```

tmrTimer 觸發時只要遞增 *FPlayingTime*，並且強制更新畫面，*DrawBackBitmap* 方法就會根據 *FPlayingTime* 的值將此關卡已花費時間畫在右上角。

tmrTitle 觸發時十分放心地呼叫 *Random* 函式取得上下左右任一方向，接著呼叫 *FRole.Move* 方法將角色移向亂數取得的方向，有點不知死活的樣子，不過這樣子不會出問題，因為我們的移動碰撞檢查碼都放在 *Move* 方法裏，所以若檢查為不合法的移動，角色就會留在原地不動，這樣一來，因為移動方向有時合法有時不合法，還可模擬出時走時停的效果呢。

tmrPlaybackTimer 的觸發事件處理方法中，將已播放的步數存在它本身的 *Tag* 屬性，然後經由 *TList* 物件 *PlaybackList* 查得目前這一步的走法，記得嗎？*TList* 物件儲放的項目型態是指標，所以要強制轉型為 *TDirection* 型態才能傳給 *TRole.Move* 方法使用。0022 列檢查，若是全部移動記錄播放完畢，則立即進入過關畫面，反正我們只有使用者過關後才會將移動記錄保留下來，因此播放的一定是過關走法，所以移動記錄全部播完畢時一定正好過關。0029 列在單元初始化時重播亂數種子，只要在程式中使用 *Random* 函式來取亂數，務必重播亂數種子，否則你將發現每回程式重新執行時所取得的亂數都一模一樣。

遊戲狀態的初始化

這些計時器由 *GameStatus* 屬性的屬性寫入方法來控制啓動狀態，*SetGameStatus* 任務重大，負責在進入各個遊戲狀態時，開關這三個計時器，並分別將該狀態所需的變數或物件初始化：

```
#0001  procedure TMainForm.SetGameStatus(Value: TGameStatus);
#0002  begin
#0003    FGameStatus := Value;
#0004
#0005    // 根據新的遊戲狀態開關三個計時器
#0006    tmrTitle.Enabled := FGameStatus = gsTitle;
#0007    tmrTimer.Enabled := FGameStatus = gsPlaying;
#0008    tmrPlayback.Enabled := FGameStatus = gsPlayback;
#0009
```

```
#0010   case FGameStatus of
#0011     gsTitle: LevelNo := 1; // 歡迎畫面顯示第一關地圖
#0012
#0013     gsPlaying:
#0014     begin
#0015       FPlayingTime := 0; // 計時歸零
#0016       FRole.CleanMoveList; // play back 歸零
#0017     end;
#0018
#0019     gsSuccess:
#0020     begin
#0021       // 過關了，將關卡記錄起來，表示要重播時就回此關卡
#0022       FPlayBackLevelNo := FLevelNo;
#0023     end;
#0024
#0025     gsPlayback:
#0026     begin
#0027       LevelNo := FPlayBackLevelNo; // 切換到記錄重播回顧的關卡
#0028       tmrPlayback.Tag := 0; // 從第一步開始播放
#0029     end;
#0030   end;
#0031
#0032   UpdateControlStatus; // 更新標題列及其它控制項
#0033   UpdateView; // 更新遊戲畫面
#0034 end;
```

因為三個計時器只有分別在歡迎畫面，遊戲中，及重播過關回顧時時才須啓動，因此 0006 ~ 0009 列根據新的遊戲狀態設定它們的 *Enabled* 屬性，同一時間最多只有一個計時器為啓動狀態。0027 列，進入重播過關回顧狀態時，必須主動載入記錄重播回顧的關卡（同時會將角色擺在初始位置），如此才可順利進行重播，要不然若目前處於第一關地圖，而動作記錄是第二關記下來的，就會看到主角到處碰壁，亂走一通的蠢模樣。

設定 *LevelNo* 屬性，也就是間接呼叫 *SetLevelNo* 方法時，裏頭再呼叫 *Map.LevelNo* 來載入關卡：

```
#0001 procedure TMainForm.SetLevelNo(Value: Integer);
#0002 begin
#0003   Map.LevelNo := Value;
#0004   FRole.X := Map.Role_X;
#0005   FRole.Y := Map.Role_Y;
#0006
#0007   FLevelNo := Value;
#0008   UpdateControlStatus;
```

```
#0009   UpdateView;
#0010  end;
```

唯一要特別注意的是，載入關卡後，千萬別忘了將角色的初始位置由 *Map* 物件的 *Role_X* 及 *Role_Y* 屬性中讀出，更新 *FRole* 的位置。

繪製遊戲畫面

有了這些幕後工作人員控制著流程，在分工清楚的前提下，前景的演員只要盡守本分，依照模式好好地繪製遊戲畫面就夠了。下列是畫出遊戲畫面的 *DrawBackBitmap* 方法：

```
#0001  procedure TMainForm.DrawBackBitmap;
#0002  var
#0003    M, S: Integer;
#0004    Str : string;
#0005    R    : TRect;
#0006  begin
#0007    Map.DrawTerrMap(FBackBitmap.Canvas); // 繪製地形層
#0008    Map.DrawItemMap(FBackBitmap.Canvas); // 繪製物品層
#0009    FRole.Draw(FBackBitmap.Canvas); // 畫出角色圖案
#0010
#0011    M := FPlayingTime div 60; // 已花費時間的分鐘數
#0012    S := FPlayingTime mod 60; // 已花費時間的秒鐘數
#0013    with FBackBitmap.Canvas do
#0014    begin
#0015      Font.Color := clWhite;
#0016      Font.Name := 'FixedSys';
#0017      Font.Style := [fsBold];
#0018      Font.Size := 14;
#0019      Brush.Style := bsClear;
#0020
#0021      case FGameStatus of
#0022        gsTitle:
#0023        begin
#0024          // 畫出上面的標題大字及下方的作者名稱
#0025          R := Rect(0, 0, TILE_WIDTH * TILE_NUM_X, TILE_HEIGHT *
#0026            TILE_NUM_Y - TextHeight('我') div 2);
#0027          DrawText(Handle, '作者: 陳寬達', - 1, R, DT_BOTTOM or
#0028            DT_CENTER or DT_SINGLELINE);
#0029          DrawStatusBox('歡迎光臨 足球番', True);
#0030        end;
#0031
```

```
#0032       gsPlayback: TextOut(5, 5,
#0033         Format('Playback (LEVEL %d) ...',[FLevelNo]));
#0034
#0035       else
#0036       begin
#0037         // 在右上角顯示時間，以 XX:XX 的格式顯示
#0038         Str := Format('%.2d:%.2d',[M, S]);
#0039         TextOut(TILE_WIDTH * TILE_NUM_X - TextWidth(Str) - 5, 5,
#0040           Str);
#0041
#0042         // 在左上角顯示關卡
#0043         Str := Format('LEVEL %d',[FLevelNo]);
#0044         TextOut(5, 5, Str);
#0045       end;
#0046     end;
#0047   end;
#0048
#0049   // 這是過關畫面
#0050   if FGameStatus = gsSuccess then
#0051     DrawStatusBox('哇，成功了 !!', False);
#0052 end;
```

0007 ~ 0009 列分別繪出地形、物品及角色，剩下的程式碼則根據目前的遊戲狀態，在畫面的不同區域畫出標題，時間，關卡及祝賀訊息等等。你會發現只要當初遊戲的狀態區分的清楚無模糊地帶，顯示畫面的程式邏輯就會簡單的不得了，根據狀態顯示不同的訊息就一切 OK。

嗯，其實已經可以看到遊戲畫面，九牛只差一毛了。還差什麼？原來居十分關鍵地位的使用者輸入處理，不能讓玩家控制的遊戲就不能叫做遊戲，而叫 DEMO 版了 :p

處理使用者輸入

首先要將 *TMainForm* 的 *KeyPreview* 屬性設定為 *True*，這樣可以保證不論鍵盤輸入焦點位於哪個控制項上，*MainForm* 本身一定會第一個收到鍵盤事件，並且還可以進行過濾處理，讓控制項本身看不到鍵盤事件哩。接著為 *TMainForm* 的 *OnKeyDown* 事件撰寫處理方法，以上下左右四個方向鍵來控制角色的移動：

```
#0001 procedure TMainForm.FormKeyDown(Sender: TObject; var Key: Word;
```

```
#0002     Shift: TShiftState);
#0003  begin
#0004    // 注意，只有遊戲中狀態，鍵盤控制才有效
#0005    if FGameStatus = gsPlaying then
#0006    begin
#0007      case Key of
#0008        VK_UP: FRole.Move(drUp); // 向上走
#0009
#0010        VK_DOWN: FRole.Move(drDown); // 向下走
#0011
#0012        VK_LEFT: FRole.Move(drLeft); // 向左走
#0013
#0014        VK_RIGHT: FRole.Move(drRight); // 向右走
#0015
#0016        else Exit; // 亂按一通，不理它
#0017      end;
#0018
#0019      UpdateView; // 更新畫面
#0020
#0021      if CheckFinished then // 是否完成任務 ??
#0022      begin
#0023        FRole.SavePlayBack; // 將行動記錄存起來以便重播
#0024        GameStatus := gsSuccess; // 進入過關狀態
#0025      end;
#0026    end;
#0027  end;
```

首先注意只有在遊戲中狀態，鍵盤控制才有效。接下來就簡單啦，檢查按鍵是否爲方向鍵，讓角色往對應的方向移動，如果不是方向鍵就離開，省得麻煩。角色更新後，呼叫 *CheckFinished* 函式檢查是否完成任務，是否已將所有目的地形利用覆蓋用物品掩住了？若是的話，就將行動記錄存起來以便重播，並且進入過關狀態。*CheckFinished* 檢查方法是這樣的：

```
#0001  function TMainForm.CheckFinished: Boolean;
#0002  var
#0003    X, Y: Integer;
#0004  begin
#0005    // 逐一檢查每個目的地形是否已被覆蓋用物品覆蓋住了 ?
#0006    for Y := 0 to TILE_NUM_Y - 1 do
#0007      for X := 0 to TILE_NUM_X - 1 do
#0008        if Map.IsTarget[X, Y] and not Map.IsSource[X, Y] then
#0009        begin
#0010          Result := False; // 哦，有一個目的地形還沒被掩住，失敗 !!
#0011          Exit;
```

```
#0012        end;
#0013
#0014    Result := True; // Yeah, 它成功了
#0015  end;
```

哇哈，它成功了，我們也成功了。再補上操作介面上的一些其它功能，遊戲主程式也完成了，迫不及待看看它的執行畫面：

圖 8-16 / 足球番的歡迎畫面

圖 8-17 / 足球番的遊戲中畫面一

圖 8-18 / 足球番的遊戲中畫面二

圖 8-19 / 足球番的過關畫面

圖 8-20 / 足球番的過關回顧畫面

圖片確實單調了些，關卡也是我隨手拉出來的，不要又打我呀，這些不是重點，遊戲寫出來才是重點咩。

你是否已有幾分感覺，撇開技術層面不談，遊戲設計與一般的程式設計其實沒有太大的差異。只要別因爲「撰寫遊戲」這四個字而興奮過頭，好好地訂立企劃，將程式中的類別、型態、模組規劃出來，再一步步慢慢兜，你將發現，遊戲程式的撰寫不但沒有想像中那麼難，所獲得的成就感還不是一般程式比得上的唷。