tUME

the Universal Map Editor

Source Code Overview

Confidential Proprietary Information

May 3, 2010

Dan Chang

Introduction

This overview of the tUME source code is designed to aid someone in modifying and enhancing tUME. It also describes enough of the workings of tUME to enable someone to add a new feature to tUME.

Building tUME

To build tUME, you need the Borland C++ 3.1 package; use the command `make -a` to invoke Borland's MAKE on the file MAKEFILE. Note that there are some environment variables that need to be set by calling the batch file `TUMEVARS.BAT`. Alternatively, use the batch file `BUILD.BAT` to delete all the old object modules, set the environment variables, and re-build tUME.

There are three header files that define compile-time switches. The file `switches.h` contains mostly debugging switches and such; for the most part we leave it alone.

The file `switch1.h` controls whether to build a demonstration version or a normal version.

The file `switch2.h` controls whether or not to include the downloading code, and which SNASM to check for.

The file `license.h` contains the tUME licensee's name.

The file `version.c` contains the version number. The version number is automatically incremented by `BUMP.EXE` everytime you build tUME.

The batch file `SAVEIT.BAT` will save the tUME source code in a zip file. The batch file `SAVETOOL.BAT` will create the zip file `tUMETOOL.ZIP` found on the tUME executables disk.

The batch file `BPIT.BAT` calls BRUSHPAK to process all the graphics in tUME (such as radio buttons, check buttons, graphic fonts, mouse pointers, etc.) to create the files `EUIBPI.BPI` and `TUMEBPI.BPI`. BRUSHPAK converts *.LBM files to some data structure that tUME understands.

The batch file `FPIT.BAT` calls FILEPACK to process the `*.BPI` files to create the file `tUME.FPF` which is loaded by `tUME.EXE`. FILEPACK collects all the various data bits used by a program, packs the data bits, and puts them together in one file.

Object Modules

tUME is built using Borland's VROOM technology. All object modules required to build tUME are listed in the MAKEFILE. There are four object module suffixes used in the MAKEFILE; they control how the object module is compiled:

`*.OBJ` ..... Compile for smallest size; also assembly language module
`*.OBF` ..... Compile for best speed
`*.OVJ` ..... Compile overlaid module for smallest size
`*.OVF` ..... Compile overlaid module for best speed

Note that all overlaid modules should be listed in the group `OVDEP`, and all non-overlaid modules should be listed in the group `PDEP`.

Main Event Loop

tUME is an event-driven program. The main loop is found in the file `tUME.c`, in the function `main()`, and looks like:

```
while (!ExitProgram) {
      ...
}
```

Inside this loop, the program calls `ReadMouse()` to read the mouse, calls `HandleMenus()` to process the menus, calls `HandleKeys()` to process any key presses, and calls `FN_DontWait()` to process mouse movements and mouse button presses on screen areas other than the menu bar.

The menus are defined in `tUME.INI`, and the code to display the menus and process the menus is found in the EGGS (Echidna Game Generation System) library `EUILIB`. When the user chooses an item on the menus, it generates the corresponding event specified in `tUME.INI`. We look up this event in the first column of `events.e` to determine the corresponding C function to call, which is listed in the second column. The menu code performs the lookup in `events.c`.; we don't need to concern ourselves with `events.c`, as it is generated by the event compiler `EVNTCOMP.EXE` when we pass `events.e` as the input file.

E.g., the user chooses **Project|Load...** to load in a tUME map. By examining the `tUME.INI` file, we determine that the event generated by this menu choice is `LoadMap`. We search the `events.e` file for the `LoadMap` event, and determine that the corresponding C function to be called is `LMap()`. Thus, when the user chooses **Project|Load...**, the event-handler function `LMap()` gets called by `HandleMenus()`.

tUME events are documented in the *tUME Configuration Guide*.

Most of the event-handler functions are found in `menuitem.c`, though a few of them are found in other modules.

Key presses are treated in a similar fashion: the keys and their corresponding events are defined in `tUME.INI`. We look up the event in `events.e` to find the corresponding event-handler function to call. E.g., the user presses `[Spacebar]`. Examining the tUME.INI file, we find that the corresponding event generated is `FlipPanes`. We search `events.e` for the corresponding event-handler for `FlipPanes`, which is `WFRoom()`. Thus, when the user pressed `[Spacebar]`, the event-handler function `WFRoom()` gets called by `HandleKeys()`.

`FN_DontWait` is a pointer to function. It may be `NULL`, or it may point at `Drawing()`, `Hovering()`, `Selecting()`, or `Tracking()`. When the user is not holding a tile-brush, `Hovering()` gets to process the mouse events; when the user is holding a tile-brush, `Tracking()` gets to process the mouse events; when the user is drag-selecting a tile-brush, `Selecting()` gets to process the mouse events; and when the user is drawing with a tile-brush, `Drawing()` gets to process the mouse events.

Initialization

The tUME.INI file is processed by `ProcessINI()` to set certain global variables, thus defining some of tUME's behavior. The events in the `[Initial Events]` section are processed by `ProcessInitialINIEvents()`. Some of the EGGS library need to be initialized as well; these initializations are performed by calling `OpenEUI()`, `OpenDBufGraphics()`, and `InitFileReqs()`.

Some sections of the tUME.INI file, such as `[Zoom Events]` and `[Cursor Movement Events]`, define new events. These events are dynamically allocated, and added to the list of events by `AddEmptyEvents()`. All events of a class point to one event handler; e.g., all zoom events point to `SetZoomEvent()`, and all cursor movement events point to `MoveCursorEvent()`. It is the responsibility of the class event handler to determine which specific zoom (or cursor, or...) event occurred.

Note that the *tUME Configuration Guide* is useful in understanding the contents of the tUME.INI file.

Data Structures

The majority of tUME's data structures are defined in `tudef.h`. We consider each object in tUME, and talk about the data structures used to represent it.

Tiles

Tiles are represented by the data structure `PlotType`, and have three attributes associated with them:

`UBYTE Plot_Flags`.. which contains flip, priority, and colorset information;
`UBYTE TileSet_ID`.. which tileset this tile belongs to; and
`UWORD Tile_ID`..... which tile in the tileset is this tile.

`TilesSet_ID`s are number starting from 1, and `Tile_ID`s are also numbered starting from 1.

Layers

A layer is represented by the data structure `LayerType`. In a linear memory machine (such as the Amiga), a layer is basically an array of `PlotType`s. However, in paged memory architectures, a layer is basically a `RGRGPLT`. A `RGRGPLT`, also defined in `tudef.h`, is basically a `MPYTMPXTPT`, along with the width and height of the layer. A `MPYTMPXTPT` is a pointer to an array of `MPXTPT`. A `MPXTPT` is just a synonym for an `XTRAPntr`. An `XTRAPntr` is a pointer to XTRA memory; see the MEMLIB docs.

The layers of a room are stored in a linked list `Layers`. The `RoomType` element `LayerType *FloorLayer` points to the node in the linked list that represents the current floor.

Layers are created by `roomio#AddLayer()`, and destroyed by `roomio#DeAllocateLayer()` and `roomio#DeAllocateLayers()`.

Rooms

A room is represented by the data structure `RoomType`. The linked list `Layers` contains the layers in the room; the structure `ColorInfo *R_ColorInfo` contains the palette information for the room.

All the rooms in tUME are stored in the linked-list contained in the data structure `MapType *GlobalMap`.

Tile-Brush

The tile-brush is represented by the data structure `BlockCopyType`. The linked list `Layers` contains the layers of the tile-brush; the elements `RoomWindowType *SourceRW`, `RoomStuffType *SourceStuff`, `RoomType *SourceRoom`, `WORD SourceX`, and `WORD SourceY` define where the tile-brush was selected.

While the user is dragging the tile-brush around, the elements `RoomWindowType *DestRW`, `RoomStuffType *DestStuff`, `WORD DestX`, and `WORD DestY` define where the tile-brush is about to be pasted.

Tilesets

A tileset is represented by the data structure `TileSetType`.

The macro `FAST_TILESET_PTR` takes a `Tileset_ID` and returns a pointer to the `TileSetType` data structure that contains information for that particular tileset ID. The functional prototype for this macro would be something like this:

```
extern TileSetType *FAST_TILESET_PTR(UBYTE TileSet_ID);
```

To convert from a `TileSetType` pointer back to a `TileSet_ID`, look up the element `WORD TS_id` in the `TileSetType` data structure.

All the tilesets in tUME are stored in an array contained in the data structure `TileSpaceType *GlobalTileSpace`.

Palettes

A palette is represented by the data structure `ColorInfo`, which is found in `colorseq.h`. Since the color information structure is fairly large (about 6K!), and since there is one for every room, they are kept in XTRA memory. The palette requester is designed to work data structure in main memory, so when it starts, it makes a copy of the colors into main memory.

Adding an Event

Here are step by step instructions for adding a new event to tUME:

1. Give the event a name, e.g., `ToggleCoolNewEvent`.

2. Add a line to represent the event in the `events.e` file, e.g.,
```
"ToggleCoolNewEvent" ToggleCoolNewEventHandler ST=CoolNewEventSTATE CHECK
TOGGLE;
```

The text in the first column in quotes is the name of this event (`"ToggleCoolNewEvent"`). The text in the second column is the name of the event handler (`ToggleCoolNewEventHandler`). This is the C function that gets called when this event is triggered. The rest of the line says that this event has a check box (`CHECK`), and that it `TOGGLE`s the state variable `CoolNewEventSTATE` between FALSE and TRUE.

3. Define the state variable in a C module. Either create a new module, or lump it into `menuitem.c`. E.g., short `CoolNewEventSTATE = FALSE;`. This sets the state variable initially FALSE.

4. Define the event handler function in a C module. Either create a new module, or lump it into `menuitem.c`. Make it a function return a short and taking no arguments, e.g., `short ToggleCoolNewEventHandler(void) { .. }`.

Look at the event `SpaceToggle` for an example of an event implement using the above four steps.

Note that it is possible to define mutually exclusive events with radio buttons in the menus. See the events `SetStampPaint` and `SetStampReplace` for an example of how to do this. If you want to define mutually exclusive events, but you don't need the radio buttons, you only need to specify the event name and the C function event handler. E.g., the events `RoomStatus`, `TileStatus`, `UserStatus`, and `CursorStatus` can be considered mutually-exclusive, but they do not have radio buttons that reflect their mutually-exclusive status.

Modules Overview

Here we examine some of the algorithms used in some of the modules of tUME.

tumedraw.c

This is the heart of tUME.

The function `StampTile()` figures out the scaling and draws the actual tile. It will also recursively call itself to draw a composite tile. The routine calls one of the assembly language routines, either `MCGA_ClippedMaskedCopyTransRect()` or `MCGA_ClippedScaledMaskedCopyTransRect()` to actually draw the tile.

Note that rooms are redrawn one layer at a time. When `Scroll()` scrolls the screen, the unchanged area is block copied, and the new area is re-drawn by calling `ShowRoomRectLayer()`.

download.c

The function `InitDownloader()` includes the calls to check for the presence of SNASM hardware.

The characters in the current layer are converted to Super Nintendo or Genesis characters. The characters are downloaded, the palette for the current room is downloaded, the map for the current layer is downloaded, and a header block defining the size of the map and other relevant information is downloaded.

The characters are collected into an array. The characters are collected by spiralling out from the current pointer position on the map. It is performed in this fashion to ensure the the map immediately surrounding the pointer utilizes the majority of the characters, and the map far away may be less detailed (as the character set may be filled by then).

If you wanted to speed up this routine, you should modify it so that it scans the layer from left to right, then from top to bottom. Activate an entire row of XTRA memory at once, the process all tiles in that row before proceeding to another row.

EGGS Library Overview

The EGGS (Echidna Game Generation System) library contains routines and data structures that are useful to the development of games on the IBM PC. The libraries are divided by function, and the library files are found in the `C:\EGGS\LIB\ECHIDNA` directory. The source for each library is found in a sub-directory with the same name (e.g., `C:\EGGS\LIB\ECHIDNA\EUILIB`). The header files for each EGGS library are found in the directory `C:\EGGS\INCLUDE\ECHIDNA`. The EGGS libraries used in tUME are:

`BEIFFLIB`..... IFF readers
`EUILIB`..... menu and keyboard support routines
`GFXLIB`..... MCGA routines
`LDSLIB`..... linked list routines
`MEMLIB`..... EMS and XMS support routines
`MISCLIB` ..... miscellaneous support routines
`TIMERLIB`..... periodic interrupt support routines
`XPAKLIB` ..... packed data loading routines

To use an EGGS library, merely link it into your program. tUME modifies some of the functionality of some of the EGGS libraries; this is accomplished by placing a local copy of the appropriate C source file from the library in the tUME directory, making the modifications to the local copy of the source file, and adding the C file to the list of modules to compile in the MAKE file.

To build an EGGS library, enter the sub-directory that contains the source code for that library, and execute the batch file `MAKEIT.BAT` found therein.

Some libraries, such as BEIFFLIB and EUILIB, are "monolithic"; they contain lots of code just to do one or two things. Other libraries, such as LDSLIB and MEMLIB, and more "granular"; they contains lots of little functions.

To understand the tUME source code, and to effectively make additions and modifications to tUME, it is important to have a firm understanding of LDSLIB and MEMLIB. Please refer to the documentation for those two EGGS libraries. While it is nice to study the other EGGS libraries as well, it is not required; study them on an "as needed" basis.

BEIFFLIB

The EGGS BEIFFLIB (Big, Easy IFF library) contains the support code necessary to load IFF ILBM files and IFF tUME files. The Big part of the name refers to the fact that this library loads the data into EMS or XMS memory provided by MEMLIB.

EUILIB

The EGGS EUILIB (Easy User Interface library) contains the support code necessary to display the windows, display the control gadgets (list boxes and elevators), display the menus, process the user's menu choices, and process the user's key presses. EUILIB also contains the standardized file requester.

GFXLIB

The EGGS GFXLIB (Graphics library) contains the support code necessary to draw on the IBM PC MCGA screen.

LDSLIB

The EGGS LDSLIB contains the linked list and binary tree support code.

MEMLIB

The EGGS MEMLIB contains the EMS and XMS memory support code.

MISCLIB

The EGGS MISCLIB contains miscellaneous support routines, such as the error reporting routines, the easy input/output routines, the INI file reading routines, the easy C strings routines, and the exit clean-up routines.

TIMERLIB

The EGGS TIMERLIB contains routines to re-program the IBM PC periodic interrupt, and provides for several concurrent interrupts at different periods.

XPAKLIB

The EGGS XPAKLIB contains routines to load FPF (FilePack'ed) files, and other support routines.

Function Locations and Brief Descriptions

The following is a list are some of the key functions in tUME and in the EGGS library, the source file where they are found, and a brief description of each function. Non-inclusion in this list does not mean the function is un-important. Use Borland's `GREP.EXE` to find other functions that are not included in the following list. Remember: `GREP` is your friend when it comes to understanding tUME.

`AddLayer()` ..... `roomio.c`: create a new layer and add it to a list of layers
`DeAllocateLayer()` `roomio.c`: free memory used by a single layer
`DeAllocateLayers()` `roomio.c`: free memory used by all layers & free layers linked-list
`Drawing()` ..... `mitems.c`: process events when user is drawing with the tile-brush
`HandleKeys()` ..... `euilib\keyevent.c`: reads keys and calls appropriate event-handler functions
`HandleMenus()` ..... `euilib\menus.c`: processes and calls appropriate event-handler functions
`Hovering()` ..... `mitems.c`: process cursor moving around with no tile-brush attached
`FN_DontWait()` ..... `mitems.c`, points to `Drawing()`, `Tracking()`, `Selecting()`, and `Hovering()`
`LMap()` ..... `menuitem.c`: loads in a tUME map
`ParseINI()` ..... `parseini.c`: read and process the tUME.INI file
`ProcessInitialINIEvents()` `parseini.c`: process `[Initial Events]` in the tUME.INI file
`ReadMouse()` ..... `ibmmouse.c`: read the mouse position and mouse buttons
`Selecting()` ..... `mitems.c`: process user drag-selecting a new tile-brush
`Scroll()` ..... `tumedraw.c`: re-draws screen when user scrolls around
`ShowRoom()` ..... `tumedraw.c`: re-draws screen to show current room
`ShowRoomRectLayer()` `tumedraw.c`: draws part of one room layer on screen
`StampTile()` ..... `tumedraw.c`: draws one properly scaled tile on screen
`Tracking()` ..... `mitems.c`: process tile-brush moving around, mouse buttons not pressed
`WFRoom()` ..... `menuitem.c`: display the other pane on screen

Compile Time Switches and Brief Descriptions

Here are some of the compile time switches used in tUME and what they mean:

`PLOTARRAY`..... defined if linear memory, currently left undefined (paged EMS & XMS memory)
`fDemoBanner`..... `switch1.h`: set to 1 if you want to build a demo version of tUME
`fDoSaveRooms` ..... `switch1.h`: set to 0 if you want to build a demo version of tUME
`dvpSNASM`..... `switch2.h`: set to 1 to include SNASM downloading code
`dvpNONE` ..... `switch2.h`: set to 1 to not include SNASM downloading code
`fCheck`..... `switch2.h`: set to fCheckNone, fCheckSNASM, fCheckGeneral, or fCheckBoth