Syndicate file format
(This is the reference for coding libsyndicate)
The libsyndicate project is in no way affiliated with Electronic Arts and/or Bullfrog Entertainment.
Syndicate and Bullfrog are trademarks of Electronic Arts.
Syndicate is © 1993 Electronic Arts.
Contents

1 Introduction 6

2 List of files and types 6
  2.1 List of data files 6
  2.2 List of data types 8

3 File format 9
  3.1 RNC 9
    3.1.1 Header 9
    3.1.2 Compressed data 9
  3.2 Palette 10
  3.3 Font 11
  3.4 Req 12
  3.5 MapData 13
  3.6 MapColumn 14
  3.7 MapTile 14
  3.8 SpriteAnim 15
  3.9 SpriteFrame 16
  3.10 SpriteElement 18
  3.11 SpriteTab 18
  3.12 SpriteData 18
  3.13 Mission 19
  3.14 Game 19
    3.14.1 Common structure 20
    3.14.2 Pedestrians 22
    3.14.3 Vehicles 25
    3.14.4 Objects 25
    3.14.5 Weapons 26
    3.14.6 Sfx 27
    3.14.7 Scenarios 28
    3.14.8 Mapinfos 28
    3.14.9 Objectives 29
  3.15 Fli 29
  3.16 Raw 30

4 Matrix of missions, maps, and games 30

5 Menus sequence 32

6 Equipement guide 32
# 7 Agents names

# 8 Methods and tools

8.1 Cheat codes ........................................ 34
8.2 Hexadecimal editor .................................. 34
8.3 Opened files ........................................ 34
8.4 Strings ............................................. 35
8.5 Memdumps .......................................... 35

# 9 References

# 10 TODO
# 1 Introduction

This is a documentation that resume informations that was needed to code the libsyndicate.

Thanks to :

- Jon Skeet for dernc.
- Andrew Sampson for the first reverse ingeniering of graphic files.
- Marcin Olak and the Desyndicate wiki.
- Stuart Binge, Joost Peters, Trent Waddington and the freesynd project staff.
- Tomasz Lis for his fan site that resume the whole above.
- Mike Melanson for details on flf files.

The author of this document is Paul Chavent (valefor@icculus.org). The Syndicate version covered by this document is the one for PC.

# 2 List of files and types

## 2.1 List of data files

This is a list of files with the type of data. Types of data are defined in section 2.2.
The prefix ‘h’ could mean Hight resolution. For example there are HPOINTER.DAT and LPOINTER.DAT.
The prefix ‘m’ could mean menu.
The prefix ‘i’ could mean intro.

<table>
<thead>
<tr>
<th>File</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL01.DAT</td>
<td>MapColumn</td>
<td>Gives the type for each of the 256 tiles</td>
</tr>
<tr>
<td>GAME[xx].DAT</td>
<td>Game</td>
<td>The description of the games</td>
</tr>
<tr>
<td>HBLK01.DAT</td>
<td>MapTile</td>
<td>The 256 base tiles that compound maps</td>
</tr>
<tr>
<td>HELE-0.ANI</td>
<td>SpriteElement</td>
<td>The descriptions of sprite elements</td>
</tr>
<tr>
<td>HFNT01.DAT</td>
<td>Font</td>
<td>FIXME : don’t know for what is it used ?</td>
</tr>
<tr>
<td>HFRA-0.ANI</td>
<td>SpriteFrame</td>
<td>The descriptions of sprite frames</td>
</tr>
<tr>
<td>HPAL01.DAT</td>
<td>Palette</td>
<td>palettes for maps</td>
</tr>
<tr>
<td>HPAL02.DAT</td>
<td>Palette</td>
<td></td>
</tr>
<tr>
<td>HPAL03.DAT</td>
<td>Palette</td>
<td></td>
</tr>
<tr>
<td>HPAL04.DAT</td>
<td>Palette</td>
<td></td>
</tr>
<tr>
<td>HPAL05.DAT</td>
<td>Palette</td>
<td></td>
</tr>
<tr>
<td>HPALETTE.DAT</td>
<td>Palette</td>
<td>FIXME</td>
</tr>
<tr>
<td>HPOINTER.DAT</td>
<td>SpriteData</td>
<td>arrow, pick, target, pointers</td>
</tr>
<tr>
<td>HPOINTER.TAB</td>
<td>SpriteTab</td>
<td></td>
</tr>
<tr>
<td>HREQ.DAT</td>
<td>GameFont</td>
<td>Fonts used for the game screen</td>
</tr>
<tr>
<td>File</td>
<td>Type</td>
<td>Comment</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>HSPr-0.DAT</td>
<td>SpriteData</td>
<td>sprites for maps</td>
</tr>
<tr>
<td>HSPr-0.TAB</td>
<td>SpriteTab</td>
<td></td>
</tr>
<tr>
<td>HSPr-0.ANI</td>
<td>SpriteAnim</td>
<td>The descriptions of sprite anims</td>
</tr>
<tr>
<td>INTRO.DAT</td>
<td>FlI</td>
<td>Animation for the introduction</td>
</tr>
<tr>
<td>INTRO.XMI</td>
<td>Music</td>
<td>Music for the introduction</td>
</tr>
<tr>
<td>ISNDS-0.DAT</td>
<td>SoundData</td>
<td>FIXME</td>
</tr>
<tr>
<td>ISNDS-0.TAB</td>
<td>SoundTab</td>
<td>FIXME</td>
</tr>
<tr>
<td>ISNDS-1.DAT</td>
<td>SoundData</td>
<td>FIXME</td>
</tr>
<tr>
<td>ISNDS-1.TAB</td>
<td>SoundTab</td>
<td>FIXME</td>
</tr>
<tr>
<td>MAP[xx].DAT</td>
<td>MapData</td>
<td>Map data (tiles reconstitution)</td>
</tr>
<tr>
<td>MBRIEOUT.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MCONFOUT.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MCONFUP.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MCONSCR.DAT</td>
<td>Raw</td>
<td></td>
</tr>
<tr>
<td>MDEBRIEOUT.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MDEOUT.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MENDLOSE.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MENDWIN.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MFNT-0.DAT</td>
<td>SpriteData</td>
<td>The menu fonts (rle encoded)</td>
</tr>
<tr>
<td>MFNT-0.TAB</td>
<td>SpriteTab</td>
<td></td>
</tr>
<tr>
<td>MGAMEWIN.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MISS[xx].DAT</td>
<td>Mission</td>
<td>Defines the mission objectives etc.</td>
</tr>
<tr>
<td>MLOGOS.DAT</td>
<td>Raw</td>
<td></td>
</tr>
<tr>
<td>MLOSA.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MLOSAOUT.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MLOSEGAM.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MMAP.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MMAPBLK.DAT</td>
<td>Raw</td>
<td></td>
</tr>
<tr>
<td>MMAPOUT.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MMINLOGO.DAT</td>
<td>Raw</td>
<td></td>
</tr>
<tr>
<td>MMULTI.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MMTOUT.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MOPTION.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MOPTOT.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MRESOUT.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MRESRCH.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MSCRRENUP.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MSELECT.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MSELECT.PAL</td>
<td>Palette</td>
<td>Palette for the menus</td>
</tr>
<tr>
<td>MSELECTOUT.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>MZPR-0.DAT</td>
<td>SpriteData</td>
<td>The menu sprites (rle encoded)</td>
</tr>
<tr>
<td>MZPR-0.TAB</td>
<td>SpriteTab</td>
<td></td>
</tr>
<tr>
<td>MTITLE.DAT</td>
<td>Fli</td>
<td></td>
</tr>
<tr>
<td>SAMPLE.AUD</td>
<td>Audio/sound</td>
<td></td>
</tr>
<tr>
<td>SAMPLE.OPN</td>
<td>Audio/sound</td>
<td></td>
</tr>
</tbody>
</table>
### Table 1: Table of files.

<table>
<thead>
<tr>
<th>File</th>
<th>Type</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOUND-0.DAT</td>
<td>SoundData</td>
<td></td>
</tr>
<tr>
<td>SOUND-0.TAB</td>
<td>SoundTab</td>
<td></td>
</tr>
<tr>
<td>SOUND-1.DAT</td>
<td>SoundData</td>
<td></td>
</tr>
<tr>
<td>SOUND-1.TAB</td>
<td>SoundTab</td>
<td></td>
</tr>
<tr>
<td>SYNGAME.XMI</td>
<td>Music</td>
<td></td>
</tr>
</tbody>
</table>

#### 2.2 List of data types

This is a list of types.

<table>
<thead>
<tr>
<th>Type</th>
<th>Reverse</th>
<th>Files associated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fli</td>
<td>INTRO.DAT INTRO.XMI MBRIEF.DAT MBRIEOUT.DAT MCONFOUT.DAT MCONF-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>FUP.DAT MCONSCR.DAT MDEBRIEF.DAT MDEOUT.DAT MENDLOSE.DAT MEND-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WIN.DAT MGAMEWIN.DAT MLOSA.DAT MLOSAOUT.DAT MLOSEGAM.DAT MMAP.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAT MMAPOUT.DAT MOPTION.DAT MOPTOUT.DAT MRESOUT.DAT MRES-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RCH.DAT MSCRENUP.DAT MSELECT.DAT MSELOUT.DAT MTITLE.DAT MMULT-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I.DAT MMULTOUT.DAT</td>
</tr>
<tr>
<td>Font</td>
<td>100%</td>
<td>HFNT01.DAT</td>
</tr>
<tr>
<td>Game</td>
<td>50%</td>
<td>GAME[xx].DAT</td>
</tr>
<tr>
<td>MapColumn</td>
<td>100%</td>
<td>COL01.DAT</td>
</tr>
<tr>
<td>MapData</td>
<td>100%</td>
<td>MAP[xx].DAT</td>
</tr>
<tr>
<td>MapTile</td>
<td>100%</td>
<td>HBLK01.DAT</td>
</tr>
<tr>
<td>Mission</td>
<td>100%</td>
<td>MISS[xx].DAT</td>
</tr>
<tr>
<td>Music</td>
<td></td>
<td>SYNGAME.XMI</td>
</tr>
<tr>
<td>Palette</td>
<td>100%</td>
<td>HPAL[xx].DAT HPALLETTE.DAT MSELECT.PAL</td>
</tr>
<tr>
<td>Raw</td>
<td>100%</td>
<td>MLOGOS.DAT MMAPBLK.DAT MMINL-OGO.DAT</td>
</tr>
<tr>
<td>Req</td>
<td>75%</td>
<td>HREQ.DAT</td>
</tr>
<tr>
<td>SoundData</td>
<td>ISNDS-[x].DAT ISNDS-[x].DAT SOUND-[x].DAT</td>
<td></td>
</tr>
<tr>
<td>SoundTab</td>
<td>ISNDS-[x].TAB SOUND-[x].TAB</td>
<td></td>
</tr>
<tr>
<td>SpriteAnim</td>
<td>100%</td>
<td>HSTA-0.ANI</td>
</tr>
<tr>
<td>SpriteFrame</td>
<td>100%</td>
<td>HFRA-0.ANI</td>
</tr>
<tr>
<td>SpriteElement</td>
<td>100%</td>
<td>HELE-0.ANI</td>
</tr>
<tr>
<td>SpriteTab</td>
<td>100%</td>
<td>HPOINTER.TAB HSPR-0.TAB MFNT-0.TAB</td>
</tr>
<tr>
<td>SpriteData</td>
<td>100%</td>
<td>HPOINTER.DAT HSPR-0.DAT MFNT-0.DAT</td>
</tr>
</tbody>
</table>
3 File format

The field are integers only, so we will use the standard int types. We prefix them with le if it’s coded in little endian, with be else. For example, if a field is a 16 bit unsigned integer, coded in little endian we call it le_uint16_t.

We will also use a base type called Block. They will be explained later.

3.1 RNC

The files are compressed with a tool called Pro-Pack from Rob Northen Computing.

There are two main parts

- an header
- the compressed data

3.1.1 Header

The header is in big endian.

```c
struct Header {
    be_uint32_t _signature;
    be_uint32_t _unpacked_length;
    be_uint32_t _packed_length;
    be_uint16_t _unpacked_crc;
    be_uint16_t _packed_crc;
    be_uint8_t _unknown;
    be_uint8_t _pack_count;
} _header;
```

The signature is always 0x524E4301 (ie the string "RNC ").

3.1.2 Compressed data

The compressed data should be read as a stream by block of 16 bits (big endian) as in the example on figure 1.

At the begining of the stream there are two unknown bits. We can skip them (don’t know what there are for now).
Then the compressed data are divided in pack. The number of pack is given by the header field _pack_count_.

Each pack begin with three huffman tables\(^1\). The structure of a table is:

- 5 bits that give the maximum value of the nodes
- 4 bits for each values that give their leaf depth

The first table is a raw table, the second is a distance table and the third is a length table. Then there is 16 bits that give the chunk count.

A chunk is (see figure 2):

- a block of raw data
- a copy of a part from a previous block

The raw length is given by the first huffman table and next bits of the stream. The raw length next bytes (aligned on 16 bits) are sent to the output.

The distance is given by the second huffman table and next bits of the stream. The distance is the offset from the current output of the pattern. We have to add 1 to the value given by the table.

The length is given by the third huffman table and next bits of the stream. The length is the length of the pattern to copy to the output. We have to add 2 to the value given by the table.

### 3.2 Palette

The Palette files are: HPAL[xx].DAT, HPALETTE.DAT, MSELECT.PAL.

Their structure is:

```c
struct Palette {
    struct Color {
        uint8_t _r;
        uint8_t _g;
        uint8_t _b;
    }
}
```

\(^1\)I like this tutorial (fr) [http://tcharles.developpez.com/Huffman/](http://tcharles.developpez.com/Huffman/)
There are 16 colours defined, and the value are between 0 and 63 (ie 7 bits).
For an 8 bits system, we need to scale the values between 0 and 255.

### 3.3 Font

The Font file is: `HFNT01.DAT`.

His structure is:

```c
struct Font {
    struct Table {
        le_uint16_t _offset;
        le_uint8_t _width;
        le_uint8_t _height;
        le_uint8_t _line_offset;
    } _tab[128];
    le_uint8_t _data[];
};
```

- `_offset` is the offset in the `_data` array,
- `_width` is the width of the font,
- `_height` is the height,
- `_line_offset` is the vertical offset where to draw the font from the top (FIXME: check),
- `_data` are the data.
The pixel are coded with one bit.
If the width is strictly less than 8, each line is coded as `le_uint8_t` where each bit represent a pixel.
If the width is greater or equal than 8, each line is coded as `le_uint16_t` where each bit represent a pixel. So the first pixel is the eightith bit, the last pixel is the seventh.

### 3.4 Req

FIXME : find a better name for this.
The Req file is: `HREQ.DAT`.
His structure is:

```c
struct Req
{
    struct Entry
    {
        Block840 _lines[_height];
        le_uint8_t _spares[16];
    } _entries[];
};
```

These data are fonts. They are 8 width and 16 height. The pixel are packed by line. So we have 16 block of 8 pixels and zero alpha channel.
A block of 8 pixel is 16 bytes. The pixels are coded on 4 bits in little endian.

- There are 32 bits for the lsb of the index of each pixels.
- There are 32 bits for the [].
- There are 32 bits for the [].
- There are 32 bits for the msb of the index of each pixels.

So they are coded as follow :

```c
struct Block840
{
    le_uint32_t _bit_0;
    le_uint32_t _bit_1;
    le_uint32_t _bit_2;
    le_uint32_t _bit_3;
};
```

So bit 0 of pixel 0 is the 7th bit of `_bit_0` and bit 0 of pixel 32 is the 24th bit of `_bit_0`.
FIXME : some fields are unknown. Moreover there isn’t alpha channel, so should we consider a value as transparent?
3.5 MapData

The MapData files are: MAP[xx].DAT.

His structure is:

```c
struct MapData {
    le_uint32_t _nb_i;
    le_uint32_t _nb_j;
    le_uint32_t _nb_k;
    le_uint32_t _offset[_nb_i * _nb_j];
    le_uint8_t _tile[?? ?? * _nb_k];
};
```

_nb_i_ is the number of tiles on i,

_nb_j_ is the number of tiles on j,

_nb_k_ is the number of tiles on k,

_offset_ is a table with the offset (from byte 12) of the tiles index,

_tile_ are the tiles index packed by stack.

For example, if we want the tile at (i;j;k), it is _tile[_offset[j * _nb_i + i] * _nb_k + k_]

The figure 3 illustrate the components of a map.

![Figure 3: A map.](image)
3.6 MapColumn

The MapColumn file is: COL01.DAT.
His structure is:

```c
struct MapColumn
{
    le_uint8_t type[256];
};
```

This file gives the type of each tiles from HBLK01.DAT:

```c
enum ColType
{
    None,
    SlopeSN,
    SlopeNS,
    SlopeEW,
    SlopeWE,
    Ground,
    RoadSideEW,
    RoadSideWE,
    RoadSideSN,
    RoadSideNS,
    Wall,
    RoadCurve,
    HandrailLight,
    Roof,
    RoadPedCross,
    RoadMark,
    NbTypes
};
```

I think that we can use this file to deduce if a tile is walkable and its color for the minimap. For example, we could say that a tile is walkable:

```c
if((None < type_k && type_k != HandrailLight && type_k < NbTypes) &&
   (None == type_k+1 || type_k+1 == HandrailLight || type_k+1 == NbTypes))
{
    tile_k is walkable
}
```

For the minimap it is not linear, it seems to be more complicated than associate a type to a colour.

3.7 MapTile

The MapTile file is: HBLK01.DAT.
His structure is:
struct MapTile
{
    le_uint32_t _offset [256][6];

    struct Subtile
    {
        Block32 _lines [16]
    } _subtile [986];
};

A tile is 64 pixels width and 48 pixels height. It is compound of 6 subtiles. Each subtile is 32 pixels width and 16 pixels height. The pixel are packed by line. So we have 16 block of 32 pixels.

A block of 32 pixel is 20 bytes. The pixels are coded on 5 bits in big-endian: one for the transparency and 4 for the index in the palette:

- There are 32 bits for the transparency bits of each pixels.
- There are 32 bits for the lsb of the index of each pixels.
- There are 32 bits for the .
- There are 32 bits for the [].
- There are 32 bits for the msb of the index of each pixels.

So they are coded as follow:

struct Block32
{
    be_uint32_t _alpha;
    be_uint32_t _bit_0;
    be_uint32_t _bit_1;
    be_uint32_t _bit_2;
    be_uint32_t _bit_3;
};

The figure 4 illustrate the components of a map tile.

### 3.8 SpriteAnim

The SpriteAnim file is: HSTA-0.ANI.

His structure is:

struct SpriteAnim
{
    le_uin16_t _indexes[];
}

This is an array of frame index. So the first frame of the 5th animation is given by _indexes[5]. Then the others frames are given in the SpriteFrame array.

The figure 5 illustrate the components of an animation.
Figure 4: Tiles, Subtiles and blocks.

### 3.9 SpriteFrame

The SpriteFrame file is: `HFRA-0.ANI`. His structure is:

```c
struct SpriteFrame
{
    struct Frames
    {
        le_uint16_t _first;
        le_uint8_t _width;
        le_uint8_t _height;
        le_uint16_t _flags;
        le_uint16_t _next;
    } _frames[];  // FIXME: give the number of frames
};
```

This is an array of frame description. A frame has a width, height, some flags and is compound of sprite elements. The index of the first sprite element is `_first`. This is an index (not an offset) in the Sprite Element tab file (see 3.10). The index of the next frame is given by `_next`. The frame index automatically loop to the first frame.

The `_flags` field seems to be 0x0100 when it is the first frame of an animation.

The frame 1450 is the persuadotron in the inventory. Then every 6 index, there is the
Figure 5: Components of an animation.
next weapon.

### 3.10 SpriteElement

The SpriteElement file is: `HELE-0.ANI`.

His structure is:

```c
struct SpriteElement
{
    struct Element
    {
        le_uint16_t _sprite;
        le_int16_t _x_offset;
        le_int16_t _y_offset;
        le_uint16_t _x_flipped;
        le_uint16_t _next;
    } _elements[]; // FIXME : give the number of elements
};
```

The `_sprite` field give the index (not an offset) in the sprite tab file (see 3.11). The `_next` is the index of the next element. If it is zero there isn’t any more elements. The `_x_flipped` attribut tells if it is horizontaly flipped.

### 3.11 SpriteTab

The SpriteTab file are: `HPOINTER.TAB`, `HSPR-0.TAB`, `MFNT-0.TAB`, `MSPR-0.TAB`.

Their structure is:

```c
struct SpriteTab
{
    struct Entry
    {
        le_uint32_t _offset;
        le_uint8_t _width;
        le_uint8_t _height;
    } _entries[]; // FIXME : give the number of entries
};
```

The offset give the number of bytes to skip from the begining of the sprite data file (see 3.12).

FIXME : give the apropiate palette for each file.

### 3.12 SpriteData

The SpriteData file are: `HPOINTER.DAT`, `HSPR-0.DAT`, `MFNT-0.DAT`, `MSPR-0.DAT`.

Their structure is:

```c
struct SpriteData
```
The sprite datas are encoded as lines of pixels (structured as block) or as rle datas. The \_nb\_sprite field gives the number of sprites. It seems that when it is not zero, the data are encoded as rle (the flag is 0x0053 for MSPR-0.DAT and 0x00CD for MFNT-0.DAT). Else, the datas are encoded as lines of pixels (structured in blocks).

If the pixel are packed by line, each line is one or more block of eight pixels. A block is 5 bytes: one for the transparency and 4 for the index in the palette.

- There are 8 bits for the transparency bits of each pixels.
- There are 8 bits for the lsb of the index of each pixels.
- There are 8 bits for the ....
- There are 8 bits for the ....
- There are 8 bits for the msb of the index of each pixels.

3.13 Mission

The Mission file are: MISSXX.DAT. They contain text and available in 4 language:

- english from 01 to 50
- french from 101 to 150
- italian from 201 to 250
- german from 301 to 350

Each string is separated with an EOL (0xa). Other separator is the pipe ‘—’ (0x7c).

3.14 Game

The Game file are: GAMEXX.DAT. Their structure is:

```c
struct GameStruct {
    le_uint8_t _header [6];
    le_uint16_t _offsets [128][128];
    le_uint16_t _offset_ref;  // (32774)
};
```
The header could be seeds for example (FIXME, not sure).
The offsets field is an array that represent the tiles of the map (every map are 128x128
tiles). The values plus 32774 give an offset in this file that is the entity placed on this
tile. The resulting offset can be 98309 max and only peds, vehicle, objects and weapons
can be indexed.
It is used for the minimap in the breifing menu. As a clue, for the first level, there are
three red points, that are in the offsets array. It is also probably (not sure) used for te
minimap in the game.
The values for offset are:

- \([2; 23554]\) pedestrian
- \([23554; 26242]\) vehicle
- \([26242; 38242]\) objects
- \([38242; 56674]\) weapons
- \([56674; 64354]\) sfx

The unkn08 field is an array of 2048 structures of 8 bytes. Perhaps it as something to
do with A.I.
The unkn11 field is an array of 129 structures of 15 bytes.
There are 116010 bytes in all files.

### 3.14.1 Common structure

The Pedestrian, Vehicle, Object and Weapon have a common header like this:

```c
struct
{
    le_uint16_t _offset_next;
    le_uint16_t _offset_prev;
    le_uint16_t _tile_i;
    le_uint16_t _tile_j;
    le_uint16_t _tile_k;
    le_uint8_t _unkn10;
    le_uint8_t _unkn11;
};
```
The _offset_prev and _offset_next plus 32774 gives the offset in this file of the previous and next entity. This is probably used for drawing the scene. I think that for drawing the scene the pseudo-algo could be:

```c
for (k = 0; k < max_k; k++)
    for (j = 0; j < 128; j++)
        for (i = 0; i < 128; i++)
            draw(tile[j * 128 + i]);
        if (_offsets[j * 128 + i] .tile_k >> 8 == k)
            entity = _offsets[j * 128 + i]
            while (entity)
                {
                    draw(entity)
                    entity = entity .offset_next
                }
```

The _tile_ give the location of the entity on the map along i, j or k. Each tile is a cube of side 256x256x128 (see fig. 3). We can deduce the tile id dividing by 256 (along i and j) or 128 (along k).

The _unkn10 is unknown but contain 0x04 for peds, 0x05 for weapons (FIXME check this).

The _index_baseAnim give an index (not an offset) in the file HSTA-0.ANI. It is the base offset for the animation of this ped.

The _index_current_frame give an index (not an offset) in the file HFRA-0.ANI. It is the current frame of the current animation displayed.

The _index_current_anim give an index (not an offset) in the file HSTA-0.ANI. It is the current animation.

The _health give the ressources of the element. For a pedestrian it will be the health, for a weapon, the amo. I am not sure it's alwas a signed int.

The _offset_unknown added to 32774 give an offset in this file. This seems to be a kind of “dependency” (see section 3.14.2).

The _type give the type of objects:

- 0x01 ped,
- 0x02 vehicle,
- 0x03 sfx,
- 0x04 weapon,
- 0x05 object.

For example, it allow to display a target or a pickup on the game screen or for the minimap. The _status may contain informations about the status of the object, or for weapons the “subtype”.

The _orientation give the initial orientation (illustrated on fig. 6) of the element:

- from 0xF0 to 0x10 : south
- from 0x10 to 0x30 : south-east
- from 0x30 to 0x50 : east
- from 0x50 to 0x70 : east-north
- from 0x70 to 0x90 : north
- from 0x90 to 0xB0 : north-west
- from 0xB0 to 0xD0 : west
- from 0xD0 to 0xF0 : west-south

Figure 6: Orientation of the ped is given in black (0x20 represent 45 degree). The offset of the animation if the ped doesn’t move is given in red. And the offset of the ped if he moves is given in blue.

### 3.14.2 Pedestrians

The _pedestrians is an array of 256 structures that describe pedestrians. This array is at address 32776 (0x8008), and each structure is 92 bytes.

```c
struct Pedestrian {
    // - 00
    le_uint16_t _offset_next;
    le_uint16_t _offset_prev;
    le_uint16_t _tile_i;
};
```
le_uint16_t tile_j;
le_uint16_t tile_k;
// - 10
le_uint8_t unk10;
le_uint8_t unk11;
le_uint8_t unk12;
le_uint8_t unk13;
le_uint16_t index_base_anim;
le_uint16_t index_current_frame;
le_uint16_t index_current_anim;
// - 20
le_int16_t health;
le_uint16_t offset_last_enemy;
le_uint8_t type;
le_uint8_t status;
le_uint16_t orientation;
le_uint8_t unk28; // when 01 pedestrian, 02 agent, 04 police, 05
le_uint8_t unk29;
// - 30
le_uint16_t unk30;
le_uint16_t offset_of_persuader;
le_uint16_t unk34;
le_uint16_t offset_of_vehicle;
le_uint16_t offset_scenario;
// - 40
le_uint16_t offset_scenario;
le_uint16_t unk42;
le_uint16_t offset_of_vehicle;
le_uint16_t goto_tile_i;
le_uint16_t goto_tile_j;
// - 50
le_uint16_t goto_tile_k;
le_uint8_t unk52[6];
le_uint16_t offset_equipment;
// - 60
le_uint16_t mods_info;
le_uint8_t unk62[6];
le_uint16_t offset_cur_weapon;
// - 70
le_uint8_t unk70;
le_uint8_t adrena_amount;
le_uint8_t adrena_dependency;
le_uint8_t adrena_effect;
le_uint8_t unk74;
le_uint8_t inteli_amount;
le_uint8_t inteli_dependency;
le_uint8_t inteli_effect;
le_uint8_t unk78;
The \verb|_unkn10| seems to be 4.

The \verb|health| of our agent can be 0x10 maximum. When it is less than zero the ped should die.

The \verb|_sub_type| is used for minimap for example (colored dots) and can be (FIXME):

- agent
- enemy agent
- criminals
- civilian
- police
- guard

The \verb|_offset_last_enemy| is the offset of the last peds that hurt, or persuade this ped (Not realy sure).

The \verb|_offset_equipment| + 32774 gives the offset in this file of the first equipment of this ped.

The \verb|_mods_info| gives the level of the mods. It its a bitfield with two bits for each

\[
\begin{array}{cccccccc}
\text{msb} & \text{spar} & \text{brain} & \text{eye} & \text{heart} & \text{chest} & \text{arm} & \text{leg} & \text{lsb} \\
\text{gender}
\end{array}
\]

The gender is 1 for female and 0 for male.

The \verb|_offset_cur_weapon| + 32774 gives the offset in this file of the current weapon in use.

The IPA levels seems to have 4 bytes. At least 3 bytes are sure. They are discribed on the picture 7.

![Figure 7: Detail of IPA field.](image)

The \verb|_offset_scenario| + 97128 gives the offset in this file of the first and/or current scenario (there are two fields).
3.14.3 Vehicles

The **vehicles** is an array of 64 structures that describe vehicles. This array is at adress 56328 (0xDC08), and each structure is 42 bytes.

```c
struct Vehicle
{
    le_uint16_t _offset_next;
    le_uint16_t _offset_prev;
    le_uint16_t _tile_i;
    le_uint16_t _tile_j;
    le_uint16_t _tile_k;
    le_uint8_t _unkn10;
    le_uint8_t _unkn11;
    le_uint8_t _unkn12;
    le_uint8_t _unkn13;
    le_uint16_t _index_base_anim;
    le_uint16_t _index_current_frame;
    le_uint16_t _index_current_anim;
    le_int16_t _health;
    le_uint16_t _offset_last_enemy;
    le_uint8_t _type;
    le_uint8_t _subtype;
    le_uint16_t _orientation;
    le_uint8_t _offset_of_ped;
    le_uint8_t _unkn30[13];
};
```

The **_offset_of_ped** + 32774 gives the offset in this file of the first ped in this vehicle.

3.14.4 Objects

The **objects** is an array of 400 structures that describe objects (trees, doors, windows, etc.). This array is at adress 59016 (0xE688), and each structure is 30 bytes.

```c
struct Object
{
    le_uint16_t _offset_next;
    le_uint16_t _offset_prev;
    le_uint16_t _tile_i;
    le_uint16_t _tile_j;
    le_uint16_t _tile_k;
    le_uint8_t _unkn10;
    le_uint8_t _unkn11;
    le_uint8_t _unkn12;
    le_uint8_t _unkn13;
    le_uint16_t _index_base_anim;
    le_uint16_t _index_current_frame;
    le_uint16_t _index_current_anim;
    le_uint8_t _unkn20[4];
};
```
The _sub_type can be:

- 0x0C door,
- 0x12 open window,
- 0x13 close window,
- 0x16 tree,
- ...

### 3.14.5 Weapons

The _weapons is an array of 512 structures that describe weapons. This array is at address 71016 (0x11568), and each structure is 36 bytes.

```c
struct Weapon {
    le_uint16_t _offset_next;
    le_uint16_t _offset_prev;
    le_uint16_t _tile_i;
    le_uint16_t _tile_j;
    le_uint16_t _tile_k;
    le_uint8_t _unkn10;
    le_uint8_t _unkn11;
    le_uint8_t _unkn12;
    le_uint8_t _unkn13;
    le_uint16_t _index_base_anim;
    le_uint16_t _index_current_frame;
    le_uint16_t _index_current_anim;
    le_uint16_t _nb_amos;
    le_uint16_t _unkn22;
    le_uint8_t _type;
    le_uint8_t _sub_type;
    le_uint16_t _orientation;
    le_uint16_t _offset_next_inventory;
    le_uint16_t _offset_prev_inventory;
    le_uint16_t _offset_owner;
    le_uint16_t _unkn34;
};
```

The _sub_type can be:

- 0x01 persuadertron,
- 0x02 pistol / air raid com,
- 0x03 gauss gun,
- 0x04 shotgun,
- 0x05 uzi,
- 0x06 minigun,
- 0x07 laser,
- 0x08 flamer,
- 0x09 long range,
- 0x0A scanner,
- 0x0B medikit,
- 0x0C time bomb,
- 0x0D access card / clone shield,
- 0x0E invalid,
- 0x0F invalid,
- 0x10 invalid,
- 0x11 energy shield.

The \texttt{nb\_amos} is the number of amos remaining. If the weapon is empty, it is equal to \texttt{0xffff} (and the weapon is not selectable). The equipment table 5 give more information about each equipment (nb amm max, range, etc.).

\textbf{FIXME} : is there any info for the picture of the weapon in the inventory ?

3.14.6 Sfx

The \texttt{sfx} is an array of 256 structures that describe sfx (for the flamer, the gauss gun, etc.). This array is at adress 89448 (0x15D68), and each structure is 30 bytes.

\begin{verbatim}
struct Sfx
{
  le_uint16_t  offset_next;
  le_uint16_t  offset_prev;
  le_uint16_t  tile_i;
  le_uint16_t  tile_j;
  le_uint16_t  tile_k;
  le_uint16_t  unkn10;
  le_uint16_t  unkn12;
  le_uint16_t  index_base_anim;
  le_uint16_t  index_current_frame;
  le_uint16_t  index_current_anim;
}
\end{verbatim}
3.14.7 Scenarios

The _scenarios is an array of 2048 structures that describe scenarios. This array is at address 97128 (0x17B68), and each structure is 8 bytes.

```c
struct Scen
{
    le Uint16_t _next;
    le Uint16_t _offset;
    le Uint8_t _i_factor;
    le Uint8_t _j_factor;
    le Uint8_t _k_factor;
    le Uint8_t _type;
};
```

The _next field gives the offset of the next point from the beginning of the structure.
The _offset field plus 32774 gives an offset in this file.
The _i_factor, _j_factor and _k_factor fields give (i,j,k) coordinates.

```c
i = (_i_factor << 7)  | 0x0040
j = (_j_factor << 7)  | 0x0040
k = (_k_factor << 7)  | 0x0000
```

3.14.8 Mapinfos

The _mapinfos is a structure that describes the map. This is at address 113960 (0x1BD28), and the structure is 14 bytes.

```c
struct Mapinfos
{
    le Uint16_t _map;
    le Uint16_t _min_x;
    le Uint16_t _min_y;
    le Uint16_t _max_x;
    le Uint16_t _max_y;
    le Uint8_t _status;
    le Uint8_t _unkn11 [3];
};
```

The _map gives the number of the map. For example, if _map is 9 the map we should open is MAP09.DAT.
The _status flag is set to 1 if the mission has been successfully completed.
3.14.9 Objectives

The _objectives is an array of 10 structures that describe the objectives of the mission. This array is at address 113974 (0x1BD36), and the structure is 14 bytes.

```c
struct Objectives {
    le_uint16_t _type;
    le_uint16_t _offset;
    le_uint16_t _tile_i;
    le_uint16_t _tile_j;
    le_uint16_t _tile_k;
    le_uint8_t _status;
    le_uint8_t _unkn11[3];
};
```

The _type field can be: 0x00 ??? ,0x01 persuade, 0x02 assassinate, 0x03 protect, 0x05 equipment acquisition, 0x0a combat sweep (police), 0x0b combat sweep, 0x0d raid and rescue, 0x0e use/destroy vehicle, 0x10 evacuate.

The _offset + 32774 gives the offset in this file of the objective.

If “protect”, the next objective are the goals and their type is zero. The list finish with zero and the offset of the protected item ? FXIME.

The _status flag is set to 1 if the objective has to be completed.

3.15 Fli

The Fli files are: INTRO.DAT, ....

There are some specific informations about Bullfrog fli files at ftp://ftp.mplayerhq.hu/MPlayer/samples/game-formats/magiccarpet-fli/.

A description of fli can be found at:

- http://www.martinreddy.net/gfx/anim/FLI.txt,
- http://steve.hollasch.net/cgindex/formats/fli.html,

Other descriptions can be found at:

- http://www.fileformat.info/format/fli/

There are some differences with the original description. The header is:

```c
struct {
    le_uint32_t _size;
    le_uint16_t _magic;
    le_uint16_t _frames;
    le_uint16_t _width;
};
```
3.16 Raw

The Raw files are: `MCONSCR.DAT`, `MLOGOS.DAT`, `MMAPBLK.DAT`, `MMINLOGO.DAT`. These files gives raw pixels.

<table>
<thead>
<tr>
<th>File</th>
<th>Nb of pictures</th>
<th>Dimension of each picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCONSCR</td>
<td>1</td>
<td>320x200</td>
</tr>
<tr>
<td>MLOGOS</td>
<td>40</td>
<td>32x32</td>
</tr>
<tr>
<td>MMAPBLK</td>
<td>50</td>
<td>64x44</td>
</tr>
<tr>
<td>MMINLOGOS</td>
<td>40</td>
<td>16x16</td>
</tr>
</tbody>
</table>

Table 3: Table of raw files.

4 Matrix of missions, maps, and games

The games from 90 to 99 are used for multiplies games. The others are the 50th.

<table>
<thead>
<tr>
<th>Games</th>
<th>Mission</th>
<th>Map</th>
<th>Palette</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>01</td>
<td>01</td>
<td>2</td>
<td>Western Europe</td>
</tr>
<tr>
<td>02</td>
<td>02</td>
<td>02</td>
<td>3</td>
<td>Far East</td>
</tr>
<tr>
<td>03</td>
<td>03</td>
<td>03</td>
<td>4</td>
<td>Mongolia</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>04</td>
<td>5</td>
<td>Iran</td>
</tr>
<tr>
<td>05</td>
<td>05</td>
<td>05</td>
<td>1</td>
<td>California</td>
</tr>
<tr>
<td>06</td>
<td>06</td>
<td>06</td>
<td>2</td>
<td>Iraq</td>
</tr>
<tr>
<td>07</td>
<td>07</td>
<td>07</td>
<td>1</td>
<td>India</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
<td>08</td>
<td>4</td>
<td>Northeast Territories</td>
</tr>
<tr>
<td>09</td>
<td>09</td>
<td>63</td>
<td>5</td>
<td>Kazakhstan</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>1</td>
<td>Eastern Europe</td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>11</td>
<td>2</td>
<td>Western Australia</td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>12</td>
<td>3</td>
<td>Kanchatka</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>13</td>
<td>4</td>
<td>Mozambique</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>20</td>
<td>5</td>
<td>Peru</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>18</td>
<td>1</td>
<td>Central Europe</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>19</td>
<td>2</td>
<td>Greenland</td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>35</td>
<td>3</td>
<td>Alaska</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>38</td>
<td>4</td>
<td>Urals</td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>34</td>
<td>5</td>
<td>Northern Territories</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>32</td>
<td>1</td>
<td>Scandinavia</td>
</tr>
<tr>
<td>Games</td>
<td>Mission</td>
<td>Map</td>
<td>Palette</td>
<td>Country</td>
</tr>
<tr>
<td>-------</td>
<td>---------</td>
<td>-----</td>
<td>---------</td>
<td>------------------</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>39</td>
<td>2</td>
<td>Yukon</td>
</tr>
<tr>
<td>22</td>
<td>22</td>
<td>41</td>
<td>3</td>
<td>Siberia</td>
</tr>
<tr>
<td>23</td>
<td>23</td>
<td>90</td>
<td>4</td>
<td>Rockies</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>70</td>
<td>5</td>
<td>Southern States</td>
</tr>
<tr>
<td>25</td>
<td>25</td>
<td>91</td>
<td>1</td>
<td>Indonesia</td>
</tr>
<tr>
<td>26</td>
<td>26</td>
<td>92</td>
<td>2</td>
<td>Mexico</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>71</td>
<td>3</td>
<td>Zaire</td>
</tr>
<tr>
<td>28</td>
<td>28</td>
<td>72</td>
<td>4</td>
<td>Algeria</td>
</tr>
<tr>
<td>29</td>
<td>29</td>
<td>40</td>
<td>5</td>
<td>New England</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>67</td>
<td>1</td>
<td>Argentine</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>93</td>
<td>2</td>
<td>South Africa</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>60</td>
<td>3</td>
<td>Colorado</td>
</tr>
<tr>
<td>33</td>
<td>33</td>
<td>61</td>
<td>4</td>
<td>Sudan</td>
</tr>
<tr>
<td>34</td>
<td>34</td>
<td>62</td>
<td>5</td>
<td>Mid West</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
<td>43</td>
<td>1</td>
<td>Lybia</td>
</tr>
<tr>
<td>36</td>
<td>36</td>
<td>31</td>
<td>2</td>
<td>Venezuela</td>
</tr>
<tr>
<td>37</td>
<td>37</td>
<td>56</td>
<td>3</td>
<td>Atlantic Accelerator</td>
</tr>
<tr>
<td>38</td>
<td>38</td>
<td>57</td>
<td>4</td>
<td>Arabia</td>
</tr>
<tr>
<td>39</td>
<td>39</td>
<td>58</td>
<td>5</td>
<td>Northwest Territories</td>
</tr>
<tr>
<td>40</td>
<td>40</td>
<td>63</td>
<td>1</td>
<td>Kenya</td>
</tr>
<tr>
<td>41</td>
<td>41</td>
<td>64</td>
<td>2</td>
<td>Mauritania</td>
</tr>
<tr>
<td>42</td>
<td>42</td>
<td>66</td>
<td>3</td>
<td>Newfoundland</td>
</tr>
<tr>
<td>43</td>
<td>43</td>
<td>65</td>
<td>4</td>
<td>New South Wales</td>
</tr>
<tr>
<td>44</td>
<td>44</td>
<td>80</td>
<td>5</td>
<td>Colombia</td>
</tr>
<tr>
<td>45</td>
<td>45</td>
<td>81</td>
<td>1</td>
<td>Nigeria</td>
</tr>
<tr>
<td>46</td>
<td>46</td>
<td>82</td>
<td>2</td>
<td>Brazil</td>
</tr>
<tr>
<td>47</td>
<td>47</td>
<td>50</td>
<td>3</td>
<td>Uruguay</td>
</tr>
<tr>
<td>48</td>
<td>48</td>
<td>51</td>
<td>4</td>
<td>Pacific Rim</td>
</tr>
<tr>
<td>49</td>
<td>49</td>
<td>53</td>
<td>5</td>
<td>Paraguay</td>
</tr>
<tr>
<td>50</td>
<td>50</td>
<td>54</td>
<td>0</td>
<td>China (Palette 0 !!! intresting !)</td>
</tr>
<tr>
<td>90</td>
<td>16</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>21</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>02</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>05</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>43</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>67</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>72</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>90</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>94</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>17</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4: Matrix of games, missions and maps.
5 Menus sequence

The menu sequence is depicted on figure 8.

![Diagram of menu sequence]

Figure 8: The succession of the different menu.

6 Equipement guide

<table>
<thead>
<tr>
<th>Item</th>
<th>Cost</th>
<th>Range</th>
<th>Ammo</th>
<th>Shot</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persuadetron</td>
<td>5000</td>
<td>256</td>
<td>- (0x0032)</td>
<td>-</td>
</tr>
<tr>
<td>Pistol</td>
<td>0</td>
<td>1280</td>
<td>13 (0x000c)</td>
<td>0</td>
</tr>
<tr>
<td>Item</td>
<td>Cost</td>
<td>Range</td>
<td>Ammo</td>
<td>Shot</td>
</tr>
<tr>
<td>------------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>Guass Gun</td>
<td>50000</td>
<td>5120</td>
<td>3 (0x0002)</td>
<td>15000</td>
</tr>
<tr>
<td>Shotgun</td>
<td>250</td>
<td>1024</td>
<td>12 (0x000b)</td>
<td>2</td>
</tr>
<tr>
<td>Uzi</td>
<td>750</td>
<td>1792</td>
<td>50 (0x0031)</td>
<td>2</td>
</tr>
<tr>
<td>Mini-Gun</td>
<td>10000</td>
<td>2304</td>
<td>500 (0x01f3)</td>
<td>10</td>
</tr>
<tr>
<td>Laser</td>
<td>35000</td>
<td>4096</td>
<td>5 (0x0004)</td>
<td>2000</td>
</tr>
<tr>
<td>Flamer</td>
<td>1500</td>
<td>512</td>
<td>1000 (0x03e7)</td>
<td>1</td>
</tr>
<tr>
<td>Long Ranger</td>
<td>1000</td>
<td>6144</td>
<td>30 (0x001d)</td>
<td>2</td>
</tr>
<tr>
<td>Scanner</td>
<td>500</td>
<td>4096</td>
<td>- (0x0013)</td>
<td>-</td>
</tr>
<tr>
<td>Medikit</td>
<td>500</td>
<td>256</td>
<td>1 (0x0001)</td>
<td>1</td>
</tr>
<tr>
<td>Time Bomb</td>
<td>25000</td>
<td>1000</td>
<td>- (0x00c7)</td>
<td>-</td>
</tr>
<tr>
<td>Access Card</td>
<td>1000</td>
<td>256</td>
<td>- (0x0000)</td>
<td>-</td>
</tr>
<tr>
<td>Energy Shield</td>
<td>8000</td>
<td>768</td>
<td>200 (0x00c7)</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 5: Table of equipements.

The \textit{nb\_amo} max for each weapon is:

- 0x01 persuadertron 0x32,
- 0x02 pistol 0x0c / air raid com,
- 0x03 gauss gun 0x02,
- 0x04 shotgun 0x0b,
- 0x05 uzi 0x31,
- 0x06 minigun 0x01f3,
- 0x07 laser 0x04,
- 0x08 flamer 0x03e7,
- 0x09 long range 0x1d,
- 0x0A scanner 0x13,
- 0x0B medikit 0x01,
- 0x0C time bomb 0xc7,
- 0x0D access card 0x00 / clone shield,
- 0x11 energy shield 0xc7.
7 Agents names

There are 68 agents:
AFSHAR AARNOLD EBAIRD LTBALDWIN BLACK BOYD PLABOYES EN BRAZIER BROWN R BUSH CARR PLACHRISMAS CLINTON COOPER ECORPES TCOX DAWSON EDONKIN TDISKET DUNNE EDGAR LAEVANS FAIRLEY FAWCETT FLINT LTFLOYD GRIFFITHS YDHARRIS EHASTINGS HERBERT HICKMAN HICKS LAHILL MASJAMES INJEFFERY JOSEPH JOHNSON JOHNSTON ONKJONES SKLEWIS NNLINDSELL LALOCKLEY MARTIN MCENTEE MCLAUGHIN OYMOLYNEUX ITHMUNRO RR MORRIS TMUMFORD NIXON PARKER PRATT LAERID MASRENNIE NRICE RIPLEY ROBERTSON HNROMANO KSEAT SKSEN SHAW SIMMONS SNELLING TAYLOR TROWSERS WEBLEY IWELLESLEY UXWILD UNRWILLIS

8 Methods and tools

8.1 Cheat codes

NUK THEM Select any country

ROB A BANK 100 million

TO THE TOP 100 million and select any country

COOPER TEAM Money and items

WATCH THE CLOCK Fast research completion

DO IT AGAIN Press [Ctrl] + C to finish mission

MARKS TEAM All country are yours

OWN THEM Select any country

8.2 Hexadecimal editor

An hexa editor is mandatory. Graphical, ones are handful when you have to explore only one file a time. For an overview of all files, command line tools are better i think. For example, to see the objectives structures:

```
for i in 'ls GAME*.DAT';
do
echo $i; od -A x -j 113974 -N 98 -t x1 —width=14 $i;
done | more
```

8.3 Opened files

To see what files are open, we can use:

```
# strace -e trace=open dosbox -conf dosbox.conf MAIN.EXE &
```
8.4 Strings
Talk abouts strings.

8.5 Memdumps
I use the Memshot tool because it seems that GAMEXX.DAT files are mapped in memory. So if we use dosbox for running Syndicate, we can inspect the dynamic of data in memory. For example:

1. - launch dosbox and enter the level 1
   
   ```
   # dosbox --conf dosbox.conf MAIN.EXE &
   ```

2. - get the pid
   
   ```
   # ps -e
   ```

3. - use Memshot
   
   ```
   # ./MemshotFe GAME01.DAT [pid [59020 [4]]]
   ```

4. - move in the game

5. - use Memshot
   
   ```
   ?> l
   ```

The game reinit if you took the shot at the begining of the level!

9 References

- Infos for rnc algorithm.
- First reverse ingenerating of graphic files.
- Desyndicate (reverse ingenerating) wiki.
- The freesynd project.
- A fan site.

10 TODO

Probably a lot of things here!