Ver. 1.4

Format plików gry Dispel

Pliki .GTL i .BTL:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Size [Byte]** |
| Tile pixels (RGB16\_565) | short[] | 1k \* 2 |
| ... |  |  |

Tile:

0

1

2

3

4

5

6

7

8

...

Width = 64

Height = 32

pos = 0;

for (int y = 0; y < tile.Height; y++)

{

var n = y < tile.Height / 2 ? y : tile.Height - 1 - y;

var r = 1 + 2 \* n;

for (int x = tile.Width / 2 - r; x < tile.Width / 2 + r; x++)

{

var byte0 = pixels[pos++];

var byte1 = pixels[pos++];

int red = byte1 & 0xF8;

int green = byte1 << 5 & 0xFF | byte0 >> 3 & 0xFC;

int blue = byte0 << 3 & 0xFF;

int alpha = red + green + blue > 0 ? 0xFF : 0;

tile[x, y] = alpha << 24 | red << 16 | green << 8 | blue;

}

}

Pliki .MAP:

Mapy z kafelków tiles są zawsze w postaci rombu o rozmiarach 2 \* HexMapSize:

Castles: 124

Dungeons: 249

Maps: 499

Final: 199

2 \* HexMapSize

MapSize

GridOffset

GridTransform

GridUntransform

Transformacja Square-Hex *GridTransform* transformuje środki komórek siatki kwadratowej [x\_,y\_] na środki komórek siatki heksagonalnej [x,y]:

i przesuwa o wektor GridOffset:

Macierz transformacji jest ortogonalna:

Jej odwrotnością jest wyskalowana (przez 1/2) macierz transponowana :

Transformacja Hex-Square *GridUntransform* cofa przesunięcie:

a następnie transformuje środki komórek siatki heksagonalnej [x,y] na środki komórek siatki kwadratowej [x\_,y\_]:

Projekt *DispelTools* używa transformacji prostej *GridTransform* do narysowania całej mapy z kafelków, a następnie obcina ją do rozmiarów MapSize i zapamiętuje na dużej bitmapie.

Zalety:

- mapa budowana jest raz, co przyspiesza rendering dla dużych pomniejszeń, ukazujących całą mapę.

Wady:

- mapa zajmuje dużo pamięci (podstawowe mapy „occluded” ok. 470MB, „non-occluded” 1.4GB)

- scena nie jest dynamiczna. Jeśli zamierzamy wyświetlać sprite’y, które mogą chować się pod kafelkami BTiles, to bitmapa może zapamiętać jedynie kafelki podłoża GTiles.

Projekt *Strategy* używa transformacji odwrotnej *GridUntransform* do narysowania mapy jedynie z kafelków i sprite’ów, które są widoczne w oknie. Nie tworzy widoku w postaci bitmapy, tylko generuje widok dynamicznie. Po wczytaniu mapy, mapa heksagonalna z kafelkami GTiles jest odtransformowywana do siatki kwadratowej, eliminując koniecznośc przeliczania współrzędnych siatki w czasie rysowania.

Zalety:

- nie tworzy bitmap i nie zużywa na nie pamięci

- scena jest dynamiczna. Pozwala wyświetlać sprite’y, które mogą chować się pod kafelkami BTiles. Podobny sposób rysowania musiał zostać zastosowany w samej grze.

- przy dużych powiększeniach rendering jest bardzo szybki.

Wady:

- przy dużych pomniejszeniach, ukazujących całą mapę, rysowanie jest nieco spowolnione.

- siatka po odtransformowaniu zawiera „dziury”. Co druga komórka w poziomie nie jest używana (związane z układem heksagonalnym siatki). Nie wpływa to oczywiście na rendering, a zużycie pamięci jest niewielkie.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Size [Byte]** |
| (HexMap.Width + 1) / 25 (HexMap.Width = HexMapSize) | int | 4 |
| (HexMap.Height + 1) / 25 (HexMap.Height = HexMapSize) | int | 4 |
| Tiles props count TPC (= 2) | int | 4 |
| GTiles.Count | int | 4 |
| GTiles props block | int[] | GTiles.Count \* TPC \* 4 |
| BTiles.Count | int | 4 |
| BTiles props block | int | BTiles.Count \* TPC |
| Sprites.Count | int | 4 |
| for (int n = 0; n < SpritesFrames.Count; n++)  spriteFrames = SpritesFrames.Add() | | |
| sprite.Stamp (= 6, 9) | int | 4 |
| sprite header | byte[] | 264 |
| frames block ID (= 8) | long | 8 |
| framesCount | long | 8 |
| for (int i = 0; i < framesCount; i++)  frame = spriteFrames.Add() | | |
| Left | int | 4 |
| Top | int | 4 |
| Right | int | 4 |
| Bottom | int | 4 |
| X (= Left, 0) | int | 4 |
| Y (= Top, 0) | int | 4 |
| Map.Left | int | 4 |
| Map.Top | int | 4 |
| width | int | 4 |
| height | int | 4 |
| pixelsCount (= width \* height) | int | 4 |
| frame.Pixels | short[] | pixelsCount \* 2 |
| endfor | | |
| UNK\_84 | byte[] | 84 |
| UNK | byte[] | (sprite.Stamp – 1) \* 364 |
| endfor | | |
| Sprites.Count | int | 4 |
| for (int n = 0; n < Sprites.Count; n++)  sprite = Sprites.Add() | | |
| framesIndex | int | 4 |
| sprite.Frames = spriteFrames[framesIndex]  for (int k = 0; k < sprite.Frames.Count; k++)  frame = sprite.Frames[k] | | |
| frame.Left | int | 4 |
| frame.Top | int | 4 |
| frame.Right | int | 4 |
| frame.Bottom | int | 4 |
| frame.X (= Left) | int | 4 |
| frame.Y (= Top) | int | 4 |
| endfor | | |
| endfor | | |
| tileObjects.Count | int | 4 |
| Block Count (=1) | int | 4 |
| for (int i = 0; i < tileObjects.Count; i++)  tileObject = tileObjects.Add() | | |
| tileObject header (= [0]) | byte[] | 260 |
| tileObject ID | int | 4 |
| frames block ID (= 8) | long | 8 |
| framesCount (= 1) | long | 8 |
| Left | int | 4 |
| Top | int | 4 |
| Right | int | 4 |
| Bottom | int | 4 |
| X (=Left) | int | 4 |
| Y (=Top) | int | 4 |
| Map.Left | int | 4 |
| Map.Top | int | 4 |
| Counter1 | int | 4 |
| Counter2 | int | 4 |
| tilesCount | int | 4 |
| for (int n = 0; n < tilesCount; n++) | | |
| tileObject.Cells[n].Piece.ImageIndex | short | 2 |
| endfor | | |
| UNK\_84 (= [0]) | byte[] | 84 |
| UNK (= [0,1]) | int[] | (Counter1 + Counter2 + tilesCount) \* 4 |
| endfor | | |
| Event block   |  |  |  | | --- | --- | --- | | EventID | short | 2 | | EventFlags (=0, 0x40(64), 0x80(128), 0xC0(192), 0x140(320), 0x280(640), 0x380(896), 0xF40(3904)) | short | 2 | | short[] | (HexMapSize^2 – 1) \* 2 \* 2 |
| GTiles block | int[] | HexMapSize^2 \* 4 |
| Roof BTiles block | int[] | HexMapSize^2 \* 4 |