Ver. 1.9

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++)

tile[x, y] = Rgb16To32(pixels[pos++], pixels[pos++]);

}

int Rgb16To32(int byte0, int byte1)

{

int color = (byte0 & 0x1F) << 3 | (byte0 & 0xE0) << 5;

color |= (byte1 & 7) << 13 | (byte1 & 0xF8) << 16;

if (color > 0) color |= unchecked((int)0xFF000000);

return color;

}

Bajty RGB (byte1, byte0) formatu piksela RGB16\_565:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| r7 | r6 | r5 | r4 | r3 | g7 | g6 | g5 | g4 | g3 | g2 | b7 | b6 | b5 | b4 | b3 |

Bajty RGB (color) formatu piksela RGB32:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| r7 | r6 | r5 | r4 | r3 | 0 | 0 | 0 | g7 | g6 | g5 | g4 | g3 | g2 | 0 | 0 | b7 | b6 | b5 | b4 | b3 | 0 | 0 | 0 |

Czwarty bajt (kanał alpha) w formacie RGB32 określa przezroczystość. Kolor czarny dla kafelków BTiles oznacza całkowitą przezroczystość (alpha = 0), pozostałe kolory są nieprzezroczyste (alpha = 0xFF).

## 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

MapWidth

GridOffset

2 \* MapHeight

GridTransform

GridUntransform

**World Space**

**Camera Space**

**Map Space**

HexMapSize x HexMapSize

2 \* HexMapSize x 2 \* HexMapSize

MapWidth x 2 \* MapHeight

MapWidth x MapHeight

, -GridOffset

GridOffset,

Y Unhex

Y Hex

Transformacja Isometric-Square *GridTransform* stanowi transformację World-Map Space, w której środki komórek siatki izometrycznej (romboidalnej) [x\_,y\_] są przeliczane na środki komórek siatki prostokątnej [x,y]. Wykonywana jest transformacja Q:

a następnie cofnięcie przesunięcia o wektor GridOffset:

i korekcja YUnHex współrzędnych Y do siatki prostokątnej:

Macierz transformacji jest ortogonalna, ponieważ jest obrotem o -45 ze skalowaniem o sqrt(2):

Jej odwrotnością jest wyskalowana (przez 1/2) macierz transponowana , czyli obrót o 45 ze skalowaniem o 1/sqrt(2):

Transformacja Square-Isometric *GridUntransform* stanowi transformację Map-World Space, która dokonuje korekcji YHex współrzędnych Y do siatki heksagonalnej:

a następnie wykonuje przesunięcie:

i transformację środków komórek siatki heksagonalnej [x’,y’] na środki komórek siatki izometrycznej [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 izometryczna z kafelkami GTiles jest odtransformowywana do siatki kwadratowej, eliminując konieczność przeliczania współrzędnych siatki w czasie rysowania i eliminując zbędne węzły w siatce.

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,int)[] | (GTiles.Count – 1) \* TPC \* 4 |
| BTiles.Count | int | 4 |
| BTiles.Props block | (byte,byte)[] | BTiles.Count \* TPC |
| Animations.Count | int | 4 |
| for (int i = 0; i < Animations.Count; i++)  animation = Animations.Add() | | |
| animation.Sequences.Count (= 6, 9) | int | 4 |
| for (int j = 0; j < animation.Sequences.Count; j++)  sequence = animation.Sequences.Add() | | |
| 0 | int | 4 |
| [0] | byte[] | 260 |
| sequence.Views.Count (= 8) | int | 4 |
| for (int k = 0; k < sequence.Views.Count; k++)  view = sequence.Views.Add() | | |
| 0 | int | 4 |
| view.Frames.Count | long | 8 |
| for (int m = 0; m < view.Frames.Count; m++)  frame = view.Frames.Add() | | |
| frame.Left | int | 4 |
| frame.Top | int | 4 |
| frame.Right | int | 4 |
| frame.Bottom | int | 4 |
| frame.X (= frame.Left, 0) | int | 4 |
| frame.Y (= frame.Top, 0) | int | 4 |
| frame.Offset.X | int | 4 |
| frame.Offset.Y | int | 4 |
| frame.Width  (= frame.Right – frame.Left) | int | 4 |
| frame.Height  (= frame.Bottom – frame.Top) | int | 4 |
| frame.Pixels.Count  (= frame.Width \* frame.Height) | int | 4 |
| frame.Pixels | short[] | frame.Pixels.Count \* 2 |
| endfor | | |
| endfor | | |
| endfor | | |
| endfor | | |
| Sprites.Count | int | 4 |
| for (int n = 0; n < Sprites.Count; n++)  sprite = Sprites.Add() | | |
| animationIndex | int | 4 |
| sprite.Animation = Animations[animationIndex]  for (int k = 0; k < sprite.Animation[0][0].Count; k++)  frame = sprite.Animation[0][0][k] | | |
| frame.Left | int | 4 |
| frame.Top | int | 4 |
| frame.Right | int | 4 |
| frame.Bottom | int | 4 |
| frame.X (= frame.Left) | int | 4 |
| frame.Y (= frame.Top) | int | 4 |
| endfor | | |
| endfor | | |
| ColumnTiles.Count | int | 4 |
| for (int i = 0; i < ColumnTiles.Count; i++)  column = ColumnTiles.Add() | | |
| 1 | int | 4 |
| [0] | byte[] | 260 |
| column ID | int | 4 |
| viewsCount (= 8) | int | 4 |
| for (int k = 0; k < viewsCount; k++) | | |
| 0 | int | 4 |
| framesCount (= 1, 0) | long | 8 |
| for (int m = 0; m < framesCount; m++) | | |
| column.Left | int | 4 |
| column.Top | int | 4 |
| column.Right | int | 4 |
| column.Bottom | int | 4 |
| column.X (= column.Left) | int | 4 |
| column.Y (= column.Top) | int | 4 |
| column cellX  (= GridUntransform(column.XY).X) | int | 4 |
| column cellY  (= GridUntransform(column.XY).Y) | int | 4 |
| 1 | int | 4 |
| column.Cells.Count | int | 4 |
| BTiles.Props.Count  (= column.Cells.Count) | int | 4 |
| for (int n = 0; n < column.Cells.Count; n++)  cell = column.Cells.Add() | | |
| cell.Piece.ImageIdx | short | 2 |
| endfor | | |
| endfor | | |
| endfor | | |
| BTiles.Props[cell.Piece.ImageIdx] | (int,int)[] | TPC \* column.Cells.Count \* 4 |
| endfor | | |
| Event block   |  |  |  | | --- | --- | --- | | EventID | short | 2 | | EventFlags (=0, 0x040(64), 0x080(128), 0x0C0(192), 0x140(320), 0x280(640), 0x380(896), 0xF40(3904)) | short | 2 | | short[] | HexMapSize^2 \* 2 \* 2 |
| GTiles block | int[] | HexMapSize^2 \* 4 |
| Roof BTiles block | int[] | HexMapSize^2 \* 4 |

Writer map umożliwia redukcję liczby pustych pętli sekwencji (z 6 lub 9 na jedną) oraz widoków view (z 8 na jedną) redukując rozmiar map w ok. 15%. Tak stworzone mapy są nadal prawidłowo interpretowane przez oryginalną grę.

## Pliki .SPR:

Zawierają blok animacji dla określonego obiektu w formacie takim samym jak dla animacji w pliku mapy (od Animations.Count do końca bloku for). W katalogach ExtraInGame, MonsterInGame i NpcInGame znajdują się animacje związane z tymi typami obiektów gry, wspólne dla wszystkich map. Pliki Extra.ini, Monster.ini i Npc.ini w katalogu głównym gry zawierają m.in. identyfikatory (indeksy) plików animacji .spr używane w plikach referencji (.ref).

## Pliki .REF:

Ich nazwa składa się z 3 pierwszych liter rodzaju zasobu (Ext, Mon lub Npc) oraz nazwy mapy i zawierają dane o obiektach zewnętrznych mapy.

Format plików „Ext\*.ref”

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Size [Byte]** |
| ElementCount | int | 4 |
| for (int i = 0; i < ElementCount; i++)  element = Elements.Add() | | |
| fileID | byte | 1 |
| Unk (=0 | byte | 1 |
| animationIndex | byte | 1 |
| name (null-terminated) | char[] | 32 |
| Type | byte | 1 |
| x | int | 4 |
| y | int | 4 |
| Sequence[0].ViewAngle | byte | 1 |
| Sequence[1].ViewAngle (=0xCD) | byte | 1 |
| Sequence[2].ViewAngle (=0xCD) | byte | 1 |
| Sequence[3].ViewAngle (=0xCD) | byte | 1 |
| Unk (=0 | int | 4 |
| SequenceIndex | int | 4 |
| required\_item1\_id (=15,4,87,13 | byte | 1 |
| required\_item1\_type (=39,4,4,4 | byte | 1 |
| required\_item1\_unk1 (=0 | byte | 1 |
| required\_item1\_unk2 (=0 | byte | 1 |
| required\_item2\_id (=15,4,87,13 | byte | 1 |
| required\_item2\_type (=39,4,4,4 | byte | 1 |
| required\_item2\_unk1 | byte | 1 |
| required\_item2\_unk2 | byte | 1 |
| Unk (=9999, | int[] | 4 \* 4 |
| gold | int | 4 |
| item1\_id (=15,46,11,2,0,4,2 | byte | 1 |
| item1\_type (=39,3,0,0,2,1,1 | byte | 1 |
| item1\_unk1 (=0 | byte | 1 |
| item1\_unk2 (=0 | byte | 1 |
| itemCount (=0|1 | int | 4 |
| Unk (=required\_item3,4? 15B,39B,0B,0B  0  15B,39B,0B,0B  0  0,0,0,0|1 | int[] | 10 \* 4 |
| eventIndex | int | 4 |
| messageIndex | int | 4 |
| Unk (=1|2B,1|2B,0B,1|2B,0xcd,0xcd,  0|1,0|1,1|0,0,1|0,(0,0,0,0|10,1,0,0)B,1; | int[] | 10 \* 4 |
| Visibility (=1 | int | 4 |
| Unk (=0 | byte[] | 3 |
| endfor | | |

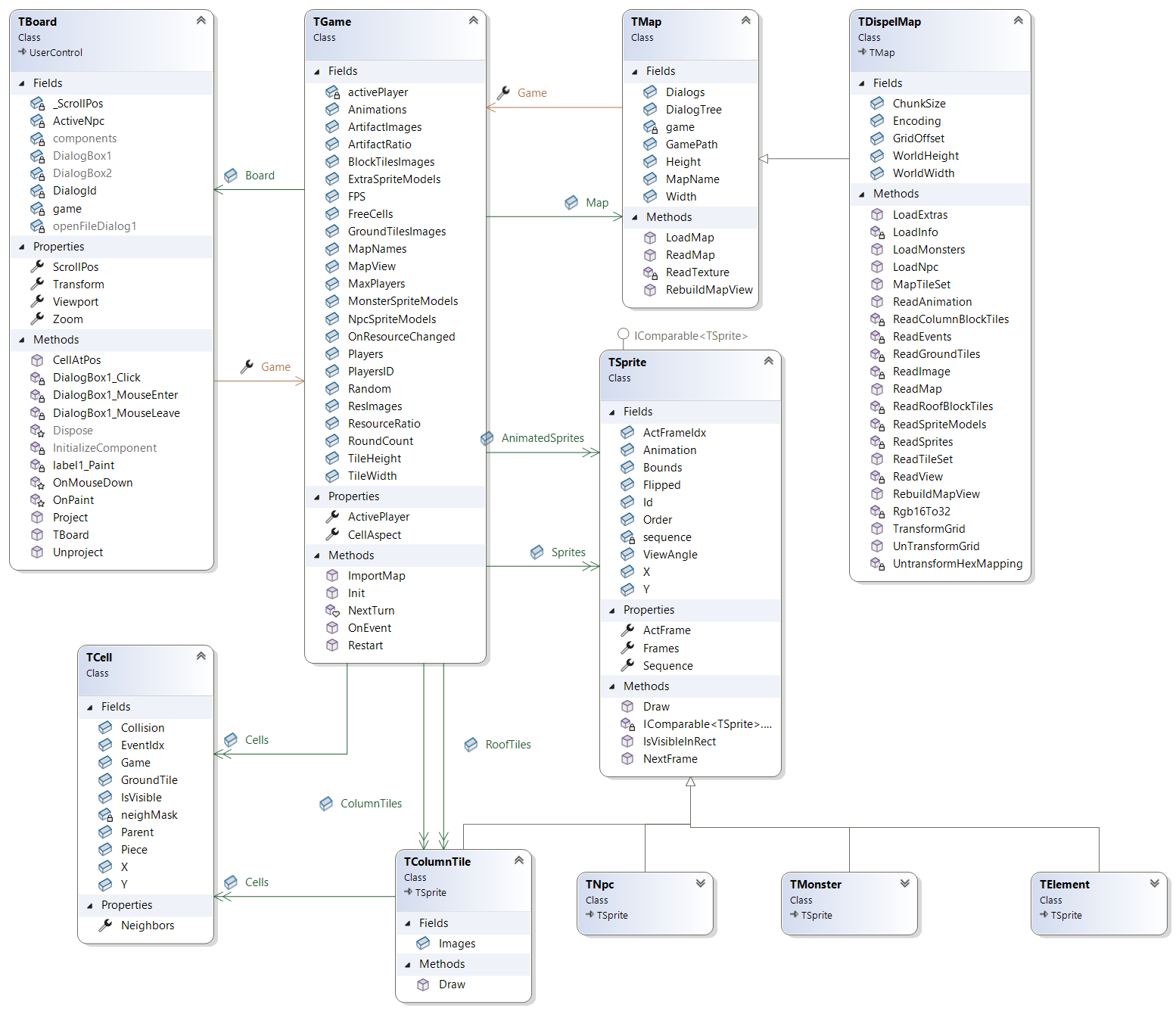
Format plików „Mon\*.ref”

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Size [Byte]** |
| ElementCount | int | 4 |
| for (int i = 0; i < ElementCount; i++)  element = Elements.Add() | | |
| fileID | int | 4 |
| animationIndex | int | 4 |
| x | int | 4 |
| y | int | 4 |
| unk | int[] | 5 \* 4 |
| lootSlot1\_id | byte | 1 |
| lootSlot1\_type | byte | 1 |
| lootSlot1\_unk1 | byte | 1 |
| lootSlot1\_unk2 | byte | 1 |
| lootSlot2\_id | byte | 1 |
| lootSlot2\_type | byte | 1 |
| lootSlot2\_unk1 | byte | 1 |
| lootSlot2\_unk2 | byte | 1 |
| lootSlot3\_id | byte | 1 |
| lootSlot3\_type | byte | 1 |
| lootSlot3\_unk1 | byte | 1 |
| lootSlot3\_unk2 | byte | 1 |
| unk1 | int | 4 |
| unk2 | int | 4 |
| endfor | | |

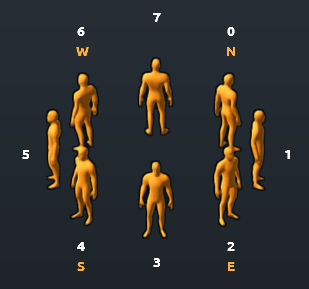
Format plików „Npc\*.ref”

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Size [Byte]** |
| ElementCount | int | 4 |
| for (int i = 0; i < ElementCount; i++)  element = Elements.Add() | | |
| fileID | int | 4 |
| animationIndex | int | 4 |
| name | char[] | 260 |
| text | char[] | 260 |
| scriptIndex | int | 4 |
| OnShowEvent | int | 4 |
| unk | int | 4 |
| Goto1\_filled | int | 4 |
| Goto2\_filled | int | 4 |
| Goto3\_filled | int | 4 |
| Goto4\_filled | int | 4 |
| Goto1X | int | 4 |
| Goto2X | int | 4 |
| Goto3X | int | 4 |
| Goto4X | int | 4 |
| Goto1Y | int | 4 |
| Goto2Y | int | 4 |
| Goto3Y | int | 4 |
| Goto4Y | int | 4 |
| unk | int[] | 4 \* 4 |
| ViewAngle | int | 4 |
| unk | int[] | 14 \* 4 |
| DialogIndex | int | 4 |
| unk | int | 4 |
| endfor | | |

## Dispel Map Viewer Class Diagram:



## Kąty widzenia (View Angles)



Rysunek . Kąty widzenia postaci

Potwory i postacie NPC mogą być przedstawiane w rzutowaniu równoległym pod 8 różnymi kątami ViewAngle, będącymi wielokrotnościami 45 stopni. Zasoby gry nie zawierają jednak animacji związanych z kierunkami 4 – 6. Są one uzyskiwane w wyniku transformacji afinicznej odbicia lustrzanego wzdłuż osi X. Klatki animacji nie muszą mieć również wspólnego punktu początkowego. Każda klatka zawiera informację o przesunięciu tego punktu względem położenia sprite’u we właściwości Offset.

int viewAngleUnflipped;

public int ViewAngle

{

get { return Flipped ? 8 - viewAngleUnflipped : viewAngleUnflipped; }

set {

viewAngleUnflipped = value;

Flipped = viewAngleUnflipped > 4;

if (Flipped)

viewAngleUnflipped = 8 - viewAngleUnflipped;

}

## }

## Ścieżki ruchu (Paths)

Ścieżki ruchu NPC są znajdowane za pomocą algorytmu Breadth-First-Search (BFS) w układzie współrzędnych świata. Każda komórka posiada właściwość uchwytu do rodzica (Parent), do którego sąsiedztwa należy dana komórka. W grze przyjęto sąsiedztwo 8-punktowe siatki (lewo, prawo, góra, dół + diagonalne). Od komórki, na której znajduje się postać, przeglądane są komórki sąsiednie i ustawiana jest właściwość Parent na komórkę startową. Następnie przeglądane są komórki sąsiednie do przejrzanych już komórek. Po odnalezieniu komórki docelowej, lista rodziców tworzy ścieżkę. Jeśli nie brać pod uwagę trudności pokonywanego terenu, można przyjąć, że wszystkie kierunki posiadają tą samą wagę krawędzi łączących komórkę z rodzicem. Połączenia diagonalne są jednak dłuższe, dlatego powinny być brane pod uwagę później, przy konstruowaniu najkrótszej ścieżki, po połączeniach na wprost.

a)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 77↘ | 37↘ | 33↓ | ↙34 | ↙44 |
| 27↘ | 7↘ | 3↓ | ↙4 | ↙04 |
| 22→ | 2→ | x | ←0 | ←00 |
| 26↗ | 6↗ | ↑1 | ↖5 | ↖05 |
| 66↗ | 16↗ | ↑11 | ↖15 | ↖55 |

b)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 55↘ | 56↓ | 56↙ | ↙67 | ↙77 |
| 45↘ | 5↘ | 6↓ | ↙7 | ↙07 |
| 35↘ | 4→ | x | ←0 | ←00 |
| 34→ | 3↗ | ↑2 | ↖1 | ↖01 |
| 33↗ | 23↗ | ↗13 | ↑12 | ↖11 |

Rysunek . Wpływ kolejności przeglądanych sąsiadów komórek na kształt ścieżki:

1. komórki na wprost przeglądane najpierw, później diagonalne
2. komórki przeglądane prawoskrętnie

## Cienie (Shadows)

Cienie potworów i postaci NPC stworzone są na podstawie oryginalnych sprite’ów na drodze transformacji afinicznych (przesunięcia, skalowania i przede wszystkim pochylenia) oraz wyzerowania intensywności oryginalnych kolorów postaci oraz nadanie im pewnego stopnia przezroczystości. W systemie graficznym Windows GDI+ transformacje afiniczne ukryte zostały w parametrze zawierającym 3 punkty definiujące docelowy równoległobok, w którym cienie wyświetlane są metodą DrawImage. Wielkość cienia oraz jego pochylenie kontrolowane są za pomocą pól *ShadowScale* i *ShadowShear*. Należy zwrócić uwagę, że dla sprite’ów odbitych chcemy uzyskać cienie odbitych postaci, ale sam cień powinien być pochylony w tym samym kierunku co postaci nieodbitych.

public override void Draw(Graphics gc)

{

var actFrame = ActFrame;

var pos = new Point(X - actFrame.Offset.X, Y - actFrame.Offset.Y);

if (Flipped) { pos.X = -X - actFrame.Offset.X; gc.ScaleTransform(-1, 1); }

var imageAttributes = new ImageAttributes();

imageAttributes.SetColorMatrix(ShadowColorMatrix);

var shadowHeight = (int)(actFrame.Bounds.Height \* ShadowScale);

var shadowShear = (int)(actFrame.Bounds.Height \* ShadowShear);

if (Flipped) shadowShear = -shadowShear;

var bounds = new Rectangle(0, 0, actFrame.Bounds.Width, actFrame.Bounds.Height);

var shadowPts = new Point[3];

shadowPts[2] = pos; shadowPts[2].Y += bounds.Height;

shadowPts[0] = shadowPts[2] + new Size(shadowShear, -shadowHeight);

shadowPts[1] = shadowPts[0]; shadowPts[1].X += bounds.Width;

gc.DrawImage(actFrame.Image, shadowPts, bounds, GraphicsUnit.Pixel, imageAttributes);

gc.DrawImage(actFrame.Image, pos.X, pos.Y);

if (Flipped) gc.ScaleTransform(-1, 1);

}

Format plików gry Diablo II

*<Tutaj opis formatów plików .dt1, .ds1, .cof i .dcc podobnie jak dla gry Dispel...>*

Animacje w Diablo II są złożone z kilku wymiennych warstw, związanych z częścią postaci podlegającej modyfikacji. Każdą z warstw stanowi lista sekwencji o tej samej liczbie klatek. Pojedyncza sekwencja definuje jeden z kilku możliwych kierunków ruchu postaci, dlatego zwykle określa się ją mianem kierunku (direction).

Wszystkie warstwy animacji można złożyć w pojedynczą animację poprzez nałożenie na siebie animacji warstw w określonej kolejności dla każdej klatki z każdego kierunku i przyspieszyć w ten sposób rendering postaci. Konieczne jest w tym celu oczywiście wyznaczenie wspólnego prostokąta obejmującego dla każdej z warstw w każdej klatce i narysowanie ich w docelowym miejscu przy uwzględnieniu offsetu klatki od tego prostokąta obejmującego.

Warstwy animacji zapisywane są w postaci skompresowanej w plikach z rozszerzeniem .dcc. Kompresja polega głównie na podziale klatek na makrobloki 4x4 pikseli i określenie dla nich palety co najwyżej 4 kolorów. Pomija się również zapis makrobloków nieruchomych podczas przejść z jednej klatki do drugiej (still blocks). Ponieważ klatki animacji nie są wyrównane do siatki prostokąta obejmującego i nie muszą mieć rozmiarów będących wielokrotnością 4, skrajne makrobloki przeważnie nie mają rozmiarów 4x4 (**Rysunek 3**).

Offset



**Rysunek 3**. Podział klatki na makrobloki (frame.Blocks) w siatce prostokąta obejmującego wszystkie klatki z danego kierunku (boundBlocks).

Każda klatka jest przesunięta względem tego prostokąta o swój wektor Offset.

Makrobloki skrajne klatki nie muszą mieć tych samych rozmiarów 4x4.