**DIABLO 2: RESURRECTED – NOTATKA**

Debugowanie i testy obecnie wykazały, że problem z wyświetleniem grafiki prawdopodobnie, leży w rozmiarach kafelków. Podkreśla to występujące trzy takie typy: *Floors and Roofs*, *Upper Walls, Shadow and Special Tiles*, *Lower Walls*.

Powoduje to, że każdy z tych typów ma oś X działającą w ten sam sposób lecz nie oś Y. Każdy blok ma własne współrzędne, które określają gdzie należy go umieścić na bitmapie. Współrzędne są zależne od typu kafelka (Tile), dlatego trzeba wiedzieć gdzie znajdują się dane współrzędne na bitmapie.

Obraz zawierający linia, zrzut ekranu, diagram, Wykres

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 1 - Floors and Roofs

Tutaj można przyjąć, że podłoga ma wymiary 160x80 pikseli (pojedynczy kafelek składający się z 25 mniejszych kafelków). Co oznacza, że wymiary to 5x5 Tile’i ( w sumie 25 Tile’i na cały).

Obraz zawierający zrzut ekranu, tekst, diagram, linia

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 2 - Upper Walls, Shadows and Special Tiles

Tutaj znane są maksymalne współrzędne (0), ale nie znamy minimalnych. Przykładem jest ***Worldstone***, który ma górną ścianę (Upper Wall Tile) o wysokości 704 piksele. Siatki podłogi są tylko orientacyjne, ponieważ górne ściany zazwyczaj nie mają własnej grafiki podłogi. Górne ściany mają na górze pusty wiersz bloków. Można więc założyć, że jeśli kafelek nie jest pusty to jego wartość jest zawyżona o 32 piksele – należy więc ją pomniejszyć o 32 przy tworzeniu bitmapy w pamięci.

Obraz zawierający zrzut ekranu, diagram, linia, kwadrat

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 3 - Lower Walls

Tutaj znamy minimalne współrzędne (-96), ale nie znamy maksymalnych. Przykładowo ***Worldstone*** ma dolną ścianę (Lower Wall Tile) o wysokości 960 pikseli. Siatka podłogi zazwyczaj nie jest tylko orientacyjna. Dolne ściany często mają tam grafikę podłogi wraz z normalnymi dolnymi ścianami.

Ważne wskazówki dotyczą przede wszystkim, że wysokość bloku nie zawsze jest wartością ujemną maksymalnej współrzędnej. Oznacza to minimalną liczbę pikseli potrzebną do narysowania kafelka. Natomiast tutaj nie ma pustych linii bloków tak jak w górnych ścianach. Dane dotyczące wysokości dokładnie odpowiadają liczbie bloków potrzebnych do stworzenia bitmapy w pamięci. Dlatego nie odejmuje się 32 od wysokości tylko rysuje zgodnie z podaną wartością.

Wcześniejsze wnioski dotyczące palety można na razie odłożyć na bok (możliwe, że potrzebna będzie potem). Obserwacje, których dokonano przy próbie rozwiązania problemu doprowadziły na pewno do tego, że zarówno plik z rozszerzeniem ***\*.dat*** oraz ***\*.pl2*** posłużą do wczytania palety barw i transformacji kolorów na kafelkch – oczywiście jak już się uda wyświetlić kafelki przynajmniej w odcieniach szarości.

Można powiedzieć, że plik ***\*.dat*** zawiera indeksy, gdzie umieścić dany kolor z palety. Sam plik zawiera ***256*** kolorów zapisanych w formacie ***RGB*** (po 3 bajty na kolor). Każdy kolor składa się z trzech wartości R, G oraz B w zakresie ***0 – 255***. Paleta używana jest do mapowania indeksów kolorów plików graficznych (***\*.dt1***) na faktyczne wartości RGB przy renderowaniu kafelków w grze.

Natomiast plik ***\*.pl2*** zawiera ***256 translacyjnych map kolorów*** (każda po 256 bajtów). Każda mapa reprezentuje ***przekształcenie*** kolorów z oryginalnej palety (0 – 255) do nowej palety (0 – 255). Używane są do efektów graficznych takich jak ***cieniowanie, oświetlenie czy efekty specjalne***. Często używane razem z plikami ***\*.dat***.

Nie wykluczone również, że problemem może być liczony ***offset***. Może być za duży dla danego pliku dlatego nie widać grafiki.

Próby zmian w ***algorytmie RLE*** doprowadziły do wniosku, że tutaj nie ma problemu z odczytywaniem danych (prócz samego PixelData równego ‘0’). Jednak nie oznacza to, że ***PixelData*** stoi na przeszkodzie rozwiązania problemu. Po zmianach w kodzie można stwierdzić, że dane są odczytywane w poprawny sposób. Wskazuje na to jeden dokonany test na podstawie pliku ***facade1.ds1*** – który jest prawdopodobnie kluczem do rozwiązania problemu wyświetlenia grafiki na kafelkach.

Na przykładzie tego pliku można było zauważyć poziome linie w odcieniach szarości (od jasnego po ciemny), oddalonych od siebie w pionie. Plik ***Bez tytułu\_white.png*** obrazuje to co jest opisane w tym fragmencie.



Rysunek 4 - facade1.ds1

Po bardzo dużym zbliżeniu za pomocą programu graficznego można zauważyć, że widoczne poziome linie to tak naprawdę piksele w odcieniach szarości. Co tylko mówi, że odcienie szarości są wyświetlane, ale nie na całym obszarze mapy.

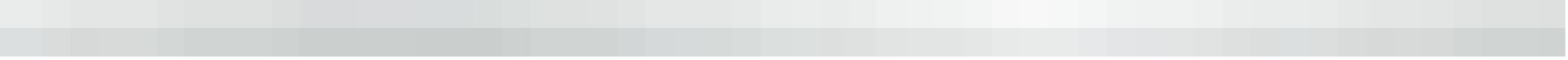
Pierwsze dwa zdjęcia ukazują pojedynczą linię odcieni szarości. Natomiast ostatni obraz pokazuje podwójną linię – jedna pod drugą).



Rysunek 5 - Odcień szarości



Rysunek 6 - Odcień szarości



Rysunek 7 - Odcienie szarości

Dalsze testy poprzez debugowanie ***pixelWritten*** oraz ***outOfBoundsCount*** pozwoliły zaobserwować i wyciągnąć wnioski. Poniżej znajduje się fragment kodu ***DecodeRLE***.

private void DecodeRLE(Block block, int w, int yOffset)

{

int pixelWritten = 0;

int outOfBoundsCount = 0;

int blockX = block.X;

int blockY = block.Y; //Math.Abs(block.Y);

int x = 0;

int y = 0;

int index = 0;

int length = block.Length;

while (length > 0)

{

byte b1 = block.EncodingData[index];

byte b2 = block.EncodingData[index + 1];

index += 2;

length -= 2;

if (b1 == 0 && b2 == 0)

{

x = 0;

y++;

continue;

}

x += (int)b1;

length -= (int)b2;

while (b2 > 0)

{

int offset = ((blockY + y + yOffset) \* w) + (blockX + x);

if (offset >= 0 && offset < block.PixelData.Length && index <  
block.EncodingData.Length)

{

//block.PixelData[offset] = block.EncodingData[index];

byte colorIndex = block.EncodingData[index];

block.PixelData[offset] = colorIndex;

block.At(x, y);

pixelWritten++;

}

else

{

outOfBoundsCount++;

}

index++;

x++;

b2--;

}

}

Console.WriteLine($"=== DecodeRLE END ===");

Console.WriteLine($"Pixels written: {pixelWritten}");

Console.WriteLine($"Out of bounds: {outOfBoundsCount}");

}

Ostatnie linijki dotyczące ***Pixel written*** oraz ***Out of Bounds*** pozwoliły zaobserwować takie wyniki jak w tabelce poniżej. Wnioski oparto o wybrane pliki map, które były testowane w trakcie debugowania.

|  |  |  |
| --- | --- | --- |
| **PLIK** | **PIXELS WRITTEN** | **OUT OF BOUNDS** |
| barewarp.ds1 | 0 | 357 |
| barnwwarp.ds1 | 0 | 64 |
| jail\_t.ds1 | 0 | 78 |
| andy3.ds1 | 0 | 344 |
| catnsew2.ds1 | 0 | 106 |
| catsew.ds1 | 0 | 106 |
| courtw.ds1 | 64 | 0 |
| cryptcountess1.ds1 | 0 | 207 |
| gravey.ds1 | 0 | 3 |
| facade1.ds1 | 255 | 0 |
| borderborder.ds1 | 0 | 70 |
| bordercliffs.ds1 | 0 | 70 |
| towne1.ds1 | 0 | 33 |
| tri\_town4.ds1 | 0 | 72 |

Tabela 1 - Wybrane pliki (Pixels Written oraz Out Of Bounds)

Uzyskane dane pozwoliły stwierdzić, że piksele są odczytywane (co na razie widać w dwóch plikach z aktu 1, ale na poparcie tego mogą być pliki znajdujące się w folderze ***expansion***– również w ich przypadku występują poziome linie).

Obraz zawierający tekst, Czcionka, pismo odręczne, dokument

Zawartość wygenerowana przez AI może być niepoprawna.

Rysunek 8 - poolroom08.ds1

Co pokazuje, że problem może leżeć w ***samych wymiarach dla kafelków*** (takie same wymiary dla wszystkich typów kafelków) – co jest błędnym założeniem i dlatego nie jesteśmy w stanie zobaczyć grafiki (ani odcieni szarości) na pozostałych mapach. Powodować to może, że ***PixelData*** jest za małe dla niektórych typów kafelków.

Poprzez anazlię ***„Same zera w PixelData”*** przez ***„Poziome paski”*** aż po ***„Problem z rozmiarem”*** – wniosek jest taki, że pliki które udało się zobaczyć w tych odcieniach szarości – zawierają jedynie kafelki *floor and roofs*. Lub w więszkości się z nich składają (zawierają mix róznych typów). Reszta nie może nawet tego uzyskać poprzez prawdopodobnie ten sam rozmiar dla wszytkich kafelków.