Mapa kafelkowa: większe kafle (sheadovas/poradniki/goto/mapa-kafelkowa-wieksze-kafle/)

Wrz 01, 2014 / goto (sheadovas/category/poradniki/goto/)

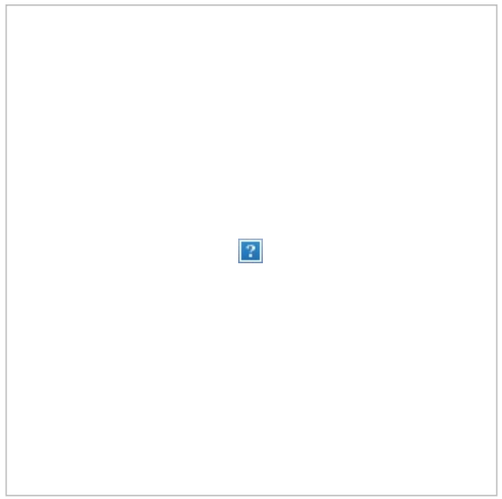
Witam Was w prezentacji/poradniku/opisie przykładowe klasy, która będzie obsługiwała wyświetlanie mapy kafelkowej, w której obiekty mogą być reprezentowane jako 1 obrazek, ale jednocześnie zajmują więcej niż 1 kafel. Będę tutaj korzystał z pseudo kodu, chociaż niewykluczone, że pojawi się jakiś kod w SFML'u (ale tylko wtedy jeżeli będziecie chcieli).

Przedstawienie problemu

Tak jak wiemy w mapie kafelkowej, mapa składa się z pojedynczych kafelków i przykładowy plik tekstowy z jego reprezentacją może wyglądać w ten sposób:

```C++
1 0 0 1
0 1 1 0
0 1 1 0
1 0 0 1
```

A po jego wyświetleniu możemy otrzymać coś takiego:



I tutaj nie mamy problemów z implementacją tego bo każda cyfra w naszym pliku tekstowym oznacza że w tym miejscu znajduje się dokładnie 1 kafelek o takich samych rozmiarach. Jednak co jeżeli posiadamy kafelek, który zajmuje miejsce np. 2 kafelków? Wtedy możemy go oczywiście podzielić na mniejsze 2 kafle, czyli traktować to jako dwa niezależne pola i w zasadzie mamy po kłopocie, jednakże musimy dodać kolejne pole do naszego kodu, czyli odpowiednik dla jednej i drugiej połowy obiektu.

Obiekt składający się z 2 kafli

Problem pojawia się przy obiektach, które składały by się z większej ilości kafli, średnio wygodnie takie kafle się ustawia bo trzeba dopasować je odpowiednio w edytorze map.

Ja poniżej przedstawię propozycję rozwiązania, które pozwoli nam na wczytanie obiektów większych niż rozmiar 1 kafla jako 1 większy obiekt bez konieczności dzielenia go na kafelki, a także bez zbędnego dzielenia go w kodzie na np lozko_gora i lozko_dol.

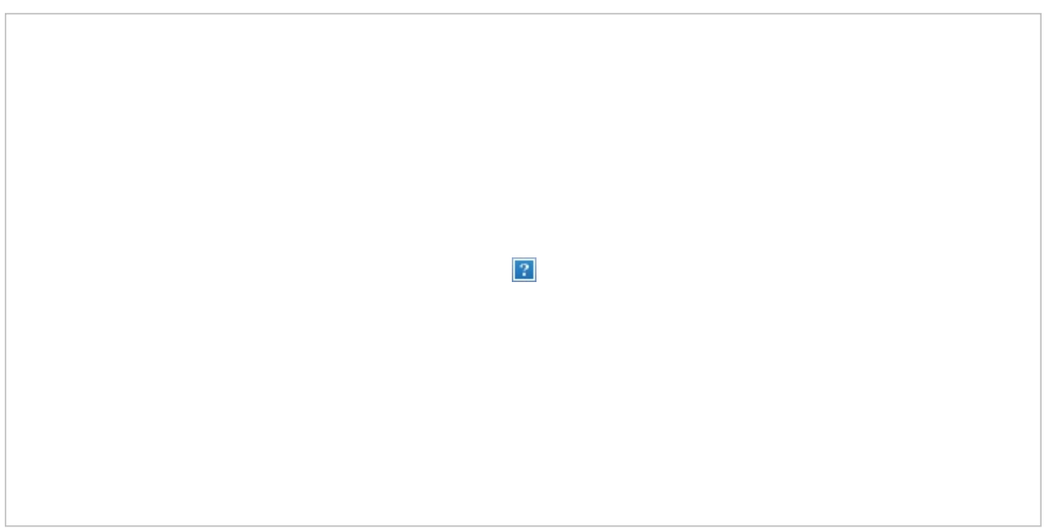
Rozwiązanie problemu

Oczywiście to w jaki sposób ja przedstawię Wam rozwiązanie tego problemu jest rozwiązaniem jednym z wielu, ja jedynie chcę Wam pokazać inne spojrzenie na ten problem.

Na początku musimy uzgodnić jakie informacje chcemy przechować w każdym kaflu. Na pewno potrzebujemy informacji o tym jakiego jest typu.

Przydałaby nam się też w nim informacja czy może nasz kafel nie jest po prostu kontynuacją innego kafla i należy go pominąć pry ustawianiu na nim tekstur.

Możemy oczywiście także dodać tutaj informacje o tym czy dany kafel jest ścianą, pułapką, etc. Czyli możemy tutaj dodać wszelkie przydatne informacje o nim potrzebne do naszej gry.



Przedstawienie zapisu wartości dla zwykłego kafla oraz obiektu o wielkości 2 kafli, które ma kolidować z graczem

```
C++
```

```cpp
class Tile
{
public:
 Typ_Kafla typ;
 bool multitile; // jezeli prawda to jest to "kontynuacja" innego kafla i pomijamy go przy ustawianiu na jego miejscu tekstury

 // opcjonalnie
 bool wall; // kafel przez ktory nie mozna przejsc
 bool trap; // pulapka ktora rani gracza
 // ...
};
```

Teraz czas na wczytanie poziomu z pliku oraz ustawieniu odpowiednich wartości dla każdego kafla. Oczywiście można to zrobić w specjalnie napisanej do tego klasie, ale my napiszemy to w zwykłej funkcji **main()**.

C++

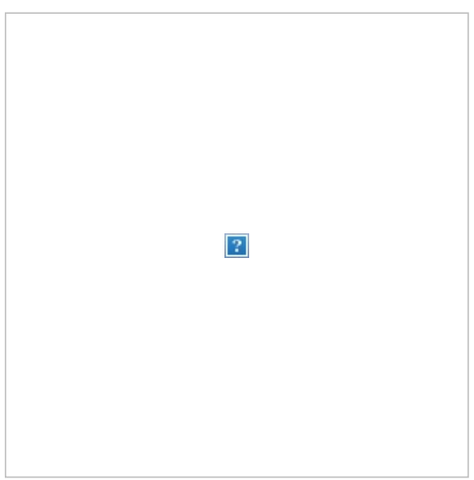
```cpp
Tile poziom[64][64]; // poziom skladajacy sie z 64 x 64 kafli

// petla wczytujaca poziom z pliku
for( int i = 0; ....)
 for( int j = 0; ...)
 {
 int tmp; // zmienna przechowujaca wczytany typ kafla
 ... // wczytanie typu kafla do 'tmp'
 poziom[i][j].typ = Typ_Kafla(tmp);
```

W kodzie powyżej wczytujemy w standardowy sposób poziom kafelkowy, który może wyglądać chociażby tak:

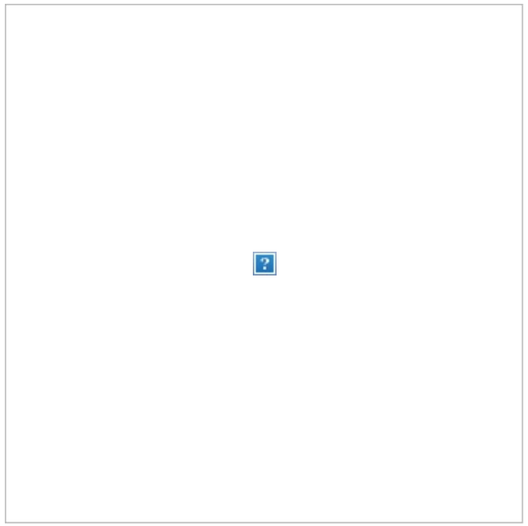
C++

```cpp
// 0 - podloga, 1 - lozko
1 0 0 0
1 0 0 0
0 0 0 0
0 0 0 0
```



Wizualizacja mapy zapisanej powyżej

Kolejnym dość ważnym krokiem jest ustawienie odpowiednio wartości dla każdego rodzaju kafla, flaga **multitile** oznacza czy ten kafel należy pominąć przy wczytywaniu tekstur tak abyśmy nie ustawili 2 razy tej samej tekstury przez co moglibyśmy otrzymać taki efekt jak na screenie poniżej, gdzie nastąpiłoby ustawienie sprite'a łóżka 2 razy, które zostałoby także zasłonięte przez kolejny kafel przy rysowaniu.



Dwukrotne wczytanie tekstury łóżka

Ostatnim krokiem jest ustawienie odpowiednich sprite’ów i wyświetlenie ich na odpowiednich pozycjach, tutaj jedyne co musimy zrobić to sprawdzić czy dany kafel ma wartość multile oznaczoną na true, jeżeli tak to pomijamy go i nie ustawiamy na nim żadnej tekstury. Resztę, czyli jego pozycję itd. ustawiamy w odpowiedni sposób dla swojej biblioteki (poradnik dla SFML (sheadovas/piszemy-gre-w-sfmlu-lekcja-3/)).

C++

```
// ustawienie odpowiednich sprite'ow
Sprite mapa[64][64]; // graficzna reprezentacja poziomu

for(...)
 for(...)
 {
 if(poziom[i][j].mutitile == true)
 break;
 else
 ... //ustawienie odpowiedniej tekstury dla danego pola
 }

window.draw(mapa);
```

Podsumowanie

Jak widzimy w dość łatwy sposób możemy napisać taką klasę, wymaga ona jedynie zrozumienia w jaki sposób są rysowane kolejne obiekty (w jakiej kolejności).

Dajcie znać co sądzicie o poradnikach pisanych przy użyciu pseudo kodu. W razie niejasności, problemów, pytań piszcie w komentarzach piszcie.

(https://github.com/sheadovas/Source-code/blob/master/inna_mapa_kafelkowa.cpp)