Wczytywanie map o niezdefiniowanych rozmiarach (sheadovas/poradniki/goto/wczytywanie-map-o-niezdefiniowanych-rozmiarach/)

Sty 30, 2015 / goto (sheadovas/category/poradniki/goto/)

W jaki sposób wczytać mapę kafelkową o niezdefiniowanym rozmiarze w kodzie.

Jest to problem z którym przy pisaniu gier dość dużo osób się spotyka, ponieważ zazwyczaj chcemy mieć możliwość posiadania grze różnych rozmiarów poziomów, a nie tylko jeden ustalony na sztywno. Sam z tym problemem się kiedyś zmierzyłem i w końcu postanowiłem nieco napisać na ten temat. Sposób jest uniwersalny, lecz my napiszemy go w oparciu o SFML (w moim wypadku SFML 2.2).

Przygotowanie

Przed właściwym rozpoczęciem programowania musimy sobie przygotować kilka rzeczy.

Tak jak wspominałem będziemy pracowali (przynajmniej ja) na SFML 2.2, oprócz tego jako bazę użyjemy znanej nam klasy *Level* (chyba, że nie czytałeś mojego poradnika Piszemy grę w SFML'u (sheadovas/piszemy-gre-w-sfmlu/)). Prezentowana poniżej wersja jest wersją finalną wspomnianego poradnika, którą zmodyfikujemy pod nasze potrzeby.

Level.h C++

```cpp
#pragma once
#include <fstream>
#include <string>

class Level
{
public:

    //funkcje skladowe klasy
    Level();
    Level(std::string filename);

    void loadFromFile(std::string filename);

    ~Level(void);

    //"podklasy"
    enum FieldType {
        NONE,
        STONE,
        DIRT,
        COUNT};

    struct Tile
    {
        FieldType type;
        bool isWall;
    };


    //zmienne
    const static int width = 32;
    const static int height = 18;
```

Level.cpp C++

```cpp
#include "Level.h"
#include <iostream>

using namespace std;

Level::Level(void)
{

}

Level::Level(std::string filename)
{
    loadFromFile(filename);
}


Level::~Level(void)
{
}

void Level::loadFromFile(std::string filename)
{
    fstream file;
    file.open("data/levels/"+filename, std::ios::in);

    if(!file.is_open())
        std::cout<<"Nie znaleziono poziomu: "+filename;
    else
    {
        for(int y=0;y<height;y++)
            for(int x=0;x<width;x++)
            {
                int tmp;
```

Podczas tego poradnika będziemy korzystali z std::vector (http://www.cplusplus.com/reference/vector/vector/), więc fajnie jakbyś chociaż mniej więcej wiedział/a na jakiej zasadzie on działa.

Wczytywanie poziomów o różnych wymiarach

Skoro mamy już bazę na której stworzymy nasz kod do wczytywania poziomów o różnych wielkościach to przejdźmy do modyfikacji kodu źródłowego:

Level.h C++

```cpp
#pragma once
#include <fstream>
#include <string>
#include <vector>    // aby używać wektorów

class Level
{
public:

    //funkcje skladowe klasy
    Level();
    Level(std::string filename);

    void loadFromFile(std::string filename);

    ~Level(void);

    //"podklasy"
    enum FieldType {
        NONE,
        STONE,
        DIRT,
        COUNT
    };

    struct Tile
    {
        FieldType type;
        bool isWall;
    };


    //zmienne
```

Oznaczone linie to linie w których dokonaliśmy zmian: dodaliśmy plik nagłówkowy, usunęliśmy przedrostki

`const static`

z rozmiarów mapy, a także zamieniliśmy tradycyjną tablicę na vectorową 2-wymiarową, gdzie deklaracja 1-wymiarowej tablicy wygląda to:

`std::vector < typ_danych > nazwa_zmiennej`

. Tego pliku już nie będziemy zmieniać i właśnie zaczyna się dopiero teraz właściwa zabawa.

```cpp
```

```cpp
void Level::loadFromFile(std::string filename)
{
    fstream file;
    file.open(filename, std::ios::in);      // w razie błędów odnośnie tej linii napisz: filename.c_str()

    if (!file.is_open()) {
        std::cout << "Nie znaleziono poziomu: " + filename;
        return;
    }

    // plik jest otwarty, wczytujemy szerokość i wysokość
    file >> width >> height;

    // powiększamy tablicę do odpowiednich rozmiarów
    poziom.resize(height);
    for (int i = 0; i < height; i++) {
        poziom[i].resize(width);
    }

    // tablica powiększona więc wczytujemy wartości
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {

            int tmp = 0;
            file >> tmp;

            poziom[y][x].type = FieldType(tmp);
            /*  ... ewentualne sprawdzenie czy ten blok koliduje z innymi,
                my to tutaj pominiemy */

            //cout << tmp << " ";
        }
        //cout << endl;
```

Przypominam, że działamy na koncepcji poziomu zbudowanego ze schematu z cyfr występujących po sobie w pliku tekstowym, tutaj z racji możliwości różnych rozmiarów dwie pierwsze cyfry to szerokość i wysokość w kaflach.

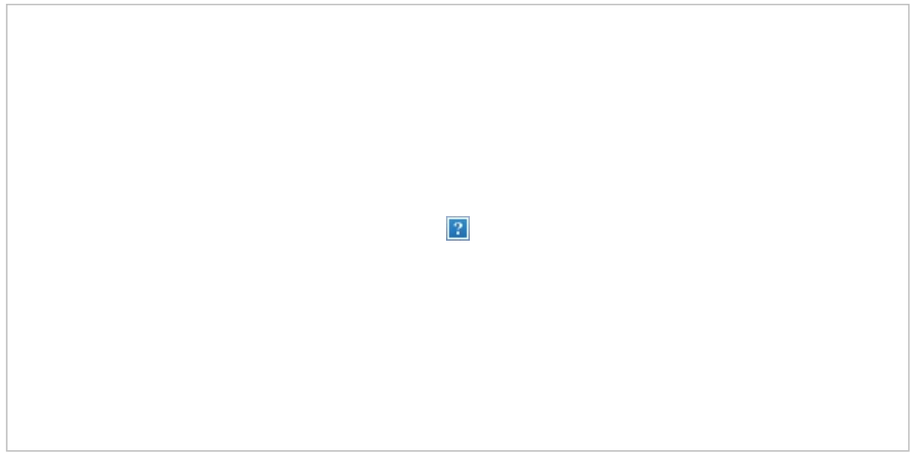
Przykładowy poziom może wyglądać w ten sposób:

Level.txt C++

```
5 5
1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1
```

W jaki sposób działa nasz kod? Otóż na początku wczytujemy `width` i `height`, które są nam potrzebne w dalszej części pracy aby odpowiednio powiększyć tablicę.

Kolejnym krokiem jest powiększenie naszej tablicy o naszą wysokość poziomu i to ona będzie pierwszą współrzędną od której będziemy używać dane, wbrew pozorom to rozwiązanie ma sens, bo nawet kafelki wczytujemy wg tej reguły: na początku idziemy do pierwszego wiersza i wczytujemy z niego wszystkie dane, później przechodzimy do kolejnego, itd. jest to jak czytanie książki, której nie czytamy od góry do dołu, tylko od lewej do prawej.



(https://i1.wp.com/sites.google.com/site/sheadovasdatabase/home/images/vector_multimap.png?ssl=1)

Następnie powiększamy każdy kolejny „case” o szerokość, jest to koniecznie, aby nasza tablica na każdym poziomie wysokości miała taką samą szerokość.

Dalszym krokiem jest wczytanie wartości poszczególnych kafli i wczytanie ich do tablicy, zwróć uwagę na kolejność w jakiej wszystko jest wykonywana (najpierw jest y, później x), zarówno w pętlach jak i tablicach `[y][x] /* a nie */ [x][y]` .

W zasadzie wszystko już mamy zrobione, pozostało nam jedynie wyświetlić to jako tablicę sprite’ów, ale nawet teraz możesz sprawdzić w konsoli poprawność wyświetlania danych (usuwając komentarze schowanego w ten sposób kodu).

Przykład użycia (SFML)

Do dalszej implementacji używam SFML i tutaj jeżeli używasz innego silnika będziesz musiał nieco pozmieniać swój kod.

```C++
```

```cpp
#include <SFMLGraphics.hpp>
#include "Level.h"

using namespace sf;

int main()
{
    // wczytujemy nasz poziom
    Level level;
    level.loadFromFile("level.txt");

    // zabawy z SFML, ja dla uproszczenia w miejsca zer będę rysował czerwony kwadrat, 1: żółty
    RectangleShape typ[2];
    for (int i = 0; i < 2; i++) {
        typ[i].setSize(Vector2f(40, 40));
        typ[i].setOutlineColor(Color(255, 102, 0));     // pomarańczowy
        typ[i].setOutlineThickness(-5);
    }
    typ[0].setFillColor(Color::Red);
    typ[1].setFillColor(Color::Yellow);

    // tworzymy tablicę o identycznych wymiarach jak w "Level", tylko że ze sprite'ami
    // lub w tym wypadku kwadratami
    std::vector <std::vector < RectangleShape > > visual;
    visual.resize(level.height);
    for (int i = 0; i < level.height; i++) {
        visual[i].resize(level.width);
    }

    // ustawiamy odpowiednie kwadraty na odpowiednich polach
    for (int i = 0; i < level.height; i++) {
        for (int j = 0; j < level.width; j++) {
            visual[i][j] = typ[level.poziom[i][j].type];
```

Myślę, że kod nie potrzebuje dodatkowego komentarza, wszystko powinno być jasne nawet dla osób, które nie piszą w SFML, w razie pytań piszcie w komentarzach.

To tyle ode mnie, mam nadzieję że poradnik okaże się przydatny, a poniżej screen z efektem działania kodu powyżej dla mapy kwadratowej oraz prostokątnej.