Totalna Kaflacja, czyli teren w grach 2D

Bartosz Szreder

Życie jest nobelon

W BTechu teren podzielony jest na hexy i występuje w 5 typach: *clear*, *rough*, *water*, *light woods*, *heavy woods*.

Ponadto teren może być na różnych wysokościach (głębokościach), ale tym się nie przejmujemy na razie.

Stan obecny: bleeding edge state-of-the-art technology, czyli jednokolorowe hexy. Jesteśmy bardziej retro i vintage niż gry na Atari 65 XL/XE.

Kontekst

BTech jest silnie podzielony na część logiczną i graficzną. Logika nie wie niczego o grafice i tak ma zostać.

- Klasa `Map` i `GraphicsMap` dziedzicząca po `Map`.
- Klasa `Hex` i `GraphicsHex` zawierająca wskaźnik na `Hex`, który reprezentuje.
- Gdzieś obok tego klasa `Grid` jako kolekcja `Hexów`, ale wiedząca o istnieniu `GraphicsHexów` i inicjująca je (pozycja na scenie).
- Jest jeszcze `GridGraphicsObject` zawierający wskaźnik do `GraphicsHex`, ale jeszcze nie wiem do czego służy...

Rysowanie

```
void GraphicsHex::paint(...)
{
    painter->setBrush(hex->getTerrain());
    painter->setPen(Qt::NoPen);
    painter->drawPath(shape());
    ...
}
```

Pobożne życzenia

Chcemy:

- rysować jakieś ładne teksturki,
- wspierać wiele różnych teksturek dla jednego rodzaju terenu, coby monotonnie nie było,
- rozsądnie obsługiwać ścieżki do obrazków w systemie plików, żeby ładowanie zasobów nie było zależne od bieżącego katalogu w momencie odpalenia programu,
- w miarę możliwości nie popsuć rozgraniczenia między logiką a grafiką.
- Bonus #1: wspierać animowane tekstury.
- Bonus #2: wspierać tekstury udające trzeci wymiar (np. „wystające” z powierzchni mapy drzewa).

The Good

Dodajemy klasę `Tile` (*kafel*), będącą lekkim opakowaniem na `QImage`. Trzymamy w środku:

- `QVector <QImage> frames` – kolejne „klatki” animowanego kafla. Nieanimowany ma tylko jedną klatkę.
- Nic więcej! Dodatkowe informacje o samym pliku graficznym (np. ścieżka) znajdują się gdzie indziej.

Warto zdecydować się na ustalony rozmiar pojedynczego kafla, np. 64×64, 256×256 etc. [/personal opinion]

Wtedy wieloklatkowe tekstury rozpoznajemy po tym, że są rozmiaru $N \times kN$.

Ponadto dodajemy do `GraphicsHex` atrybut `const Tile *tile`.

Rysowanie lepiej

```
void GraphicsHex::paint(...)
{
    painter->setBrush(hex->getTerrain());
    painter->setPen(Qt::NoPen);
    painter->drawPath(shape());
    if (tile != nullptr)
        painter->drawImage(QRect(...),
                           tile->currentFrame());
    ...
}
```

Metoda currentFrame() zwraca właściwą klatę w oparciu o globalny licznik połączony z timerem.

Ładowanie zasobów z systemu plików – RFC

- Jest taka przydatna metoda (statyczna): `QCoreApplication::applicationDirPath()`.
- Ustalamy w projekcie położenie różnych zasobów (np. podkatalog `data/tiles/`) i ładujemy je korzystajac z powyższej metody.
- Trzeba uważać na ścieżki bezwzględne. Zrobiłem sobie taką pomocniczą funkcyjkę:

```
QString resolvePath(const QString &path)
{
    if (QDir::isAbsolutePath(path))
        return path;
    return QCoreApplication::applicationDirPath()
                + '/' + path;
}
```

install target smaczny i zdrowy

- Trzeba jeszcze upewnić się, że zasoby są tam, gdzie być powinny.
- Dodajemy do `CMakeLists.txt`:

```
install (DIRECTORY data DESTINATION "${BTech_BINARY_DIR}")
```

- `make && make install`

```
QDir tileDir(BTech::resolvePath(...));
QFileInfoList imgFileList
    = tileDir.entryInfoList({"*.png"},
                    QDir::Files | QDir::Readable);

for (QFileInfo imgFile : imgFileList) {
    const Tile *tile =
            TileManager::registerTile(imgFile);
    if (tile != nullptr)
        tiles_.append(tile);
}
```

The Bad – `TileManager`

Klasa (singleton) skupiająca w sobie informacje o:

- wszystkich plikach graficznych (na potrzeby edytora map),
- pamiętaniu które pole ma który kafel (na potrzeby serializacji).

Więcej mapowań się nie dało?

- `QHash <QString, const Tile *> pathToTile;`
- `QHash <const Tile *, QFileInfo> tileToFileInfo;`
- `QVector <Tile *> tiles;`
- `QHash <QPair <int, int>, const Tile *> coordToTile;`

Międzymordzie

- `static QString fileName(const Tile *tile);`
- `static void loadTileDictionary(QDataStream &in, const QVector <Hex *> &hexes);`
- `static const Tile * registerTile(const QFileInfo &tileFile);`
- `static void saveTileDictionary(QDataStream &out, const QVector <Hex *> &hexes);`
- `static const Tile * tile(QPair <int, int> hexCoord);`

Ładowanie jednego obrazka

```
const Tile * TileManager::loadTileImage
             (const QString &filePath)
{
    QImage image(filePath);
    if (!image.isNull()
            && image.height() == Tile::TileSize
            && image.width() % Tile::TileSize == 0) {
        Tile *result = new Tile(image);
        tiles.append(result);
        pathToTile[filePath] = result;
        tileToFileInfo[result] = filePath;
        return result;
    }

    return nullptr;
}
```

Przyszedł raz Breżniew w dzień do fryzjera

```
const Tile * TileManager::registerTile
             (const QFileInfo &tileFile)
{
    TileManager &manager = instance();

    const QString filePath = tileFile.filePath();
    if (manager.pathToTile.contains(filePath))
        return manager.pathToTile[filePath];

    return manager.loadTileImage(filePath);
}
```

Zapisując mapę, dopisujemy na jej koniec:

- Słownik `UID`→`QString path`, zawierajacy tylko obrazki używane na danej mapie. `UID`y generujemy w locie.
- Odwzorowanie ze współrzędnej `Hexa` do `UID` kafla (dla `Hex`ów posiadających kafle różne od `nullptr`).

```
void TileManager::saveTileDictionary(QDataStream &out,
        const QVector <Hex *> &hexes)
{
    TileManager &manager = instance();

    UID tileCnt = EmptyUid;
    QHash <const Tile *, UID> tileToId;
    QHash <UID, QString> tileDict;

    manager.coordToTile.clear();
    for (const Hex *hex : hexes) {
        const GraphicsHex *graphicsHex
                    = GraphicsFactory::get(hex);
        const Tile *tile = graphicsHex->getTile();
```

```
if (tile != nullptr) {
    QPoint hexCoord = hex->getPoint();
    UID tileUid;
    if (tileToId.contains(tile)) {
        tileUid = tileToId[tile];
    } else {
        tileUid = ++tileCnt;
        tileToId[tile] = tileUid;
    }

    manager.coordToTile[
        qMakePair(hexCoord.x(), hexCoord.y())] = tile;
    tileDict[tileUid] =
        manager.tileToFileInfo[tile].filePath();
}
```

```
QHash <QPair <int, int>, UID> coordToTileUid;
for (auto i = manager.coordToTile.constBegin();
          i != manager.coordToTile.constEnd(); ++i)
    coordToTileUid[i.key()] = tileToId[i.value()];
out << tileDict << coordToTileUid;
```

Modrzew

```
void TileManager::loadTileDictionary(QDataStream &in,
        const QVector <Hex *> &hexes)
{
    TileManager &manager = instance();

    QHash <UID, QString> tileDict;
    QHash <UID, const Tile *> idToTile;
    QHash <QPair <int, int>, UID> coordToTileUid;
    in >> tileDict >> coordToTileUid;

    for (auto i = tileDict.constBegin();
            i != tileDict.constEnd(); ++i) {
        const Tile *tile = registerTile(i.value());
        if (tile != nullptr)
            idToTile[i.key()] = tile;
    }
```

Chyba działa

```
manager.coordToTile.clear();
for (auto i = coordToTileUid.constBegin();
          i != coordToTileUid.constEnd(); ++i)
    manager.coordToTile[i.key()]
          = idToTile[i.value()];
```

Po czymś takim mamy załadowany słownik `coordToTile` i możemy go wykorzystywać do odpowiadania na zapytania o kafel dla konkretnej współrzędnej.

```
const Tile * TileManager::tile
        (QPair <int, int> hexCoord)
{
    return instance()
            .coordToTile.value(hexCoord, nullptr);
}
```

TODO

Jeden bonus będzie kłopotliwy.

- Wsparcie dla pseudo-trójwymiarowych grafik.
- Wymaga odejścia od ustalonego na sztywno rozmiaru kafla.
- Trzeba dodać informację o przesunięciu kafla podczas rysowania albo mieć dodatkowe założenia o rozmiarze kafla (np. zawsze kwadratowe per klatka).