Heroes of dungeons and classes: map editor

Введение

Ваша задача -- написать редактор карт для игры **Heroes of dungeons and classes**.

Пользователь создает карту, добавляет на нее элементы ландшафта, монстров и героев. В любой момент можно нарисовать то, что сейчас получилось. Выглядит это как-то так:

```
> map 20 10
> show
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . .
> create lake depth 3 at 1 1 size 3
> create mountain height 1000 at 15 7 size 2
> create forest at 9 1 size 2
> create hero Axe at 1 1 hp 500
> create monster at 7 1 hp 100
> show
A . . . . . m . . . . . . . . . . T T
. O O O . . . . . . . . . . . . . T T
. O O O . . . . . . . . . . . . . . .
. O O O . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . ^ ^ . . . .
. . . . . . . . . . . . . ^ ^ . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
> dump
Dumping 5 objects:
1. Clean lake of depth 3 at (1,1) size 3
2. Big mountain 1000 feet high at (15,7) size 2
3. Forest at (9,1) size 2
4. Great hero Axe with hp 500 at (1,1)
5. Some monster with hp 100 at (7,1)
> move 4 6 1
> show
. . . . . A m . . . . . . . . . . T T
. O O O . . . . . . . . . . . . . T T
. O O O . . . . . . . . . . . . . . .
. O O O . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . ^ ^ . . . .
. . . . . . . . . . . . . ^ ^ . . . .
. . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . .
> quit
Bye!
```

Команды

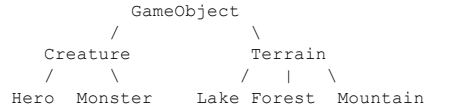
Основная программа в цикле ждет команды от пользователя (печатая перед каждой командой приглашение `"> "`). Команды бывают следующие:

- `map` создает новую карту, указываются ширина и высота
- `show` выводит текущее состояние карты на экран
- `create` создает игровой объект. Это составная команда, в зависимости от того, *что* мы создаем, она в дальнейшем принимает разные аргументы (см. пример)
- `dump` выводит описание всех имеющихся на карте объектов в текстовом виде (см. пример), вызывая `GameObject::print()`
- `move` перемещает объект (аргументом указывается индекс объекта и новое положение)
- `help` выводит инструкцию по командам
- `quit` выходит из редактора

При желании вы можете изменить синтаксис команд для собственных нужд.

Реализация

Т.к. мы изучаем объекто-ориентированное программирование в c++, то структура игрового движка должна подчиняться определенным правилам. У нас будет следующая иерархия классов:



Класс **GameObject** выглядит как-то так (это набросок, и в нем явно не все методы указаны):

```
class GameObject
{
  public:
    GameObject(int x, int y, int size);
    void move(int new_x, int new_y);    // переместить объект в новую позицию
    virtual char symbol() const;        // символ, которым отрисовывается объект, переопределяется у дочерних классов;
    virtual void print() const;         // вывести на экран описание, переопределяется у дочерних классов
  private:
    int x, y;       // координаты
    int size;       // размер
};
```

Каждый объект (экземпляр класса **GameObject**) изображается квадратом с левым верхнем углом `(x, y)` и стороной `size` с помощью символа `symbol()`. У существ **Creature** есть *очки здоровья* `hp` (*hit points*), а их размер (`size`) всегда равен единице. У героев (`Hero`) есть *имя* (`std::string`). Все нетерминальные классы (т.е. те, у которых есть потомки -- **GameObject**, **Creature**, **Terrain**) должны здесь возвращать `'?'` в `symbol()`, подчеркивая, что их экземпляры создаваться и отрисовываться не должны (позже мы изучим *абстрактные классы* с *чисто виртуальными функциями*, и тогда их создать в принципе не получится). Герои отрисовываются первой буквой имени, монстры -- буквой `'m'`, озеро -- `'O'`, гора -- `'^'`, лес -- `'F'`. Можете придумать свои способы, если они вам нравятся больше.

Также у **GameObject** есть метод `virtual void print() const`, выводящий описание объекта в текстовом виде (используется в команде `dump`).

`class World` будет отвечать за мир в целом. Главный метод класса **World** -- `void render() const`, печатающий на экран (в стандартный поток вывода) текущее состояние карты. **World** будет содержать в себе указатели на вновь созданные объекты, а в деструкторе удалять их. Примерный вид класса:

```
class World
{
  public:
    World(int width, int height);      // инициализирует все поля, вызывает render() для создания пустой карты
    ~World();   // удаляет картину миры и все объекты, т.е. проходится по ``objects`` и для каждого элемента вызывает ``delete``

    void show() const;    // выводит на экран текущую картину мира (``std::cout << map;``)
    void add_object(GameObject* new_object);    // добавляет указатель на вновь созданный объект в массив ``objects`` и вызывает
                                                // ``render()``

  private:
    void render();    // заполняет картину мира (``map``), но не отрисовывает ее. Вызывается каждый раз после ``add_object``
    int width, height;

    GameObject* objects[MAX_GAME_OBJECTS];  // массив указателей на объекты. Те, кто умеют обращаться с ``std::vector``, пусть
                                            // используют его
    int objects_count;    // текущее количество объектов. Если вместо ``objects`` используется ``std::vector``, эта переменная
                          // не нужна

    char* map;       // C-строка размера ``height*(width*2+1)+1``  (подумайте, почему именно такого),
                     //содержащая в себе полную картину мира (т.е. все переводы строк уже включены)

    World(const World&);   // запретить копирование (любители c++11 могут использовать ``=delete``)
    void operator= (const World&);  // запретить присваивание (любители c++11 могут использовать ``=delete``)
};
```

Проверка

Программу необходимо проверить на наличие утечек памяти. В Microsoft Visual Studio для этого в файле, где определена `main` можно добавить строки

```
#define _CRTDBG_MAP_ALLOC
#include <stdlib.h>
#include <crtdbg.h>

class AutoLeakChecker
{
  public:
    ~AutoLeakChecker()
    {
      _CrtDumpMemoryLeaks();
    }
} global_leak_checker;
```

Запускать через *Start with debugging* и после выхода смотреть на *Output window*.

TODO

Вот, что можно было бы добавить в следующие версии:

- запретить объектам пересекаться (в одной клетке сейчас может быть очень много объектов)
- использовать `std::vector` для массива объектов (чтобы не было фиксированного верхнего предела по количеству)
- отслеживать коллизии имен: героя могли назвать *Osiris*, и его изображение путали бы с озером, также не должно быть героев с одинаковой первой буквой имени
- автодополнение команд: работать, если пользователь ввел часть (префикс) существующей команды и он однозначно определяет ее (например, `d` для `dump`)
- обращаться к объектам не по индексу, а по имени (для команды `move`), например, `move Axe 30 40` (для героя Axe) или `move lake 50 60` (если на карте только одно озеро)
- дать возможность указывать относительные перемещения, например, `move Axe +3 +4` для перемещения относительно текущей позиции
- сделать классы **GameObject**, **Creature** и **Terrain** *интерфейсами* (т.е. чисто виртуальными)
- сохранение мира в файл и чтение из файла