```
/*
 Name: Michael Campbell
 Date: 6/22/2011
 File: game.cpp
 */

#include <iostream>
#include <sstream>
#include "except.h"
#include "game.h"
#include "manual.h"
#include "immobile.h"
#include "brownian.h"
#include "rank.h"
#include "grandchild.h"
using namespace std;

game::game(char initial_c)
: term(initial_c)
{
    static const size_t xmax = 36;        //number of columns in the picture
    static const char a[][xmax + 1] = {   //plus 1 for terminating '\0'
        "....................................",
        "....................................",
        "......bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb",
        "......b............................b",
        "......b.....r..............s.......b",
        "......b............................b",
        "......b............................b",
        "......b.....r......................b",
        "......b..............W.............b",
        "......b............................b",
        "......bbbbbbbbbbbbbb.bbbbbbbbbbbbbbb"
    };
    static const size_t ymax = sizeof a / sizeof a[0];

    try {
        for (size_t y = 0; y < ymax; ++y) {
            for (size_t x = 0; x < xmax; ++x) {
                if (term.in_range(x, y)) {
                    switch (a[y][x]) { //sorry y before x
                        case '.':
                            break;

                        case 'b': //boulder
                            typedef grandchild<immobile, inert_t, 'b'>
                                boulder_t;
                            new boulder_t(this, x, y);
                            break;

                        case 'r': //rabbit
                            typedef grandchild<brownian, victim_t, 'r'>
                                rabbit_t;
                            new rabbit_t(this, x, y);
                            break;

                        case 's': //sitting_duck
```

```
                                typedef grandchild<immobile, victim_t, 's'>
                                    sitting_duck_t;
                                new sitting_duck_t(this, x, y);
                                break;

                            case 'W': //wolf
                                typedef grandchild<manual, predator_t, 'W'> wolf_t;
                                new wolf_t(this, x, y);
                                break;

                            default:
                                ostringstream os;
                                os << "bad character '"
                                << a[y][x]
                                << "' at (" << x << ", "
                                << y << ")\n";
                                throw except(os);
                        }
                    }
                }
            }
        }

        catch (...) {
            depopulate();
            throw;
        }
    }

void game::depopulate()
{
    for (master_t::const_iterator it = master.begin(); it != master.end();){
        const wabbit *const p = *it;
        ++it;
        delete p;
    }
}

game::master_t::value_type game::get(unsigned x, unsigned y) const
{
    for (master_t::const_iterator it = master.begin(); it != master.end();
         ++it) {
        const master_t::value_type p = *it;
        if (p->x == x && p->y == y) {
            return p;
        }
    }

    return 0;
}

game::master_t::size_type game::count(char c) const
{
    master_t::size_type n = 0;

    for (master_t::const_iterator it = master.begin(); it != master.end();
         ++it) {
```

```
        if ((*it)->c == c) {
            ++n;
        }
    }

    return n;
}

void game::play()
{
    for (;; term.wait(250)) {
        for (master_t::const_iterator it = master.begin();
             it != master.end();) {

            wabbit *const p = *it;
            const bool alive = p->move();
            ++it;

            if (!alive) {
                //The wabbit that just moved blundered into
                //another wabbit and was eaten.
                delete p;
            }

            if (count('s') == 0 && count('r') == 0) {
                term.put(0, 0, "You killed all the "
                         "sitting ducks and rabbits.");

                //Give user three seconds to read the message.
                term.wait(3000);

                return;
            }
        }
    }
}
```

```
/*
 Name: Michael Campbell
 Date: 6/22/2011
 File: rank.h
 */

#ifndef RANKH
#define RANKH

#include <climits>
#include "wabbit.h"

template <int HUNGRY, int BITTER>
class rank: private virtual wabbit {
    int hungry() const {return HUNGRY;}
    int bitter() const {return BITTER;}
public:
    rank(game *initial_g, unsigned initial_x, unsigned initial_y, char
        initial_c)
    :wabbit(initial_g, initial_x, initial_y, initial_c) {}
};

//Convenient names for the rank classes:

typedef rank<INT_MIN, INT_MAX> inert_t;
typedef rank<INT_MIN, INT_MIN> victim_t;
typedef rank<INT_MAX, INT_MAX> predator_t;
typedef rank<INT_MAX, INT_MIN> halogen_t;

#endif
```

```
/*
 Name: Michael Campbell
 Date: 6/22/2011
 File: grandchild.h
 */

#ifndef GRANDCHILDH
#define GRANDCHILDH

//MOTION must have member functions decide and (optionally) punish;
//RANK must have member functions hungry and bitter.

template <class MOTION, class RANK, char C>
class grandchild: private MOTION, private RANK {
public:
    grandchild(game *initial_g, unsigned initial_x, unsigned initial_y)
    :wabbit(initial_g, initial_x, initial_y, C),
     MOTION(initial_g, initial_x, initial_y, C),
     RANK(initial_g, initial_x, initial_y, C)
    {}
};

#endif
```