```
#ifndef TERMINALH
#define TERMINALH

#include <iostream>
#include <sstream>
#include "except.h"
#include "terminal.h"
using namespace std;

extern "C" {
#include "term.h"
}

template <class CHAR = char>
class terminal {
    const CHAR _background;   //default value for third argument of put
    const unsigned _xmax;     //number of columns of characters
    const unsigned _ymax;     //number of rows of characters

    void check(unsigned x, unsigned y) const;
public:
    terminal(CHAR initial_background = ' ');
    ~terminal();

    typedef CHAR value_type;

    CHAR background() const {return _background;}
    unsigned   xmax() const {return _xmax;}
    unsigned   ymax() const {return _ymax;}

    void put(unsigned x, unsigned y, CHAR c) const;
    void put(unsigned x, unsigned y) const {put(x, y, _background);}
    CHAR get(unsigned x, unsigned y) const {check(x, y); return term_get(x, y);}

    char key() const {return term_key();}
    void wait(int milliseconds) const {term_wait(milliseconds);}
    void beep() const {term_beep();}

    bool in_range(unsigned x, unsigned y) const {return x < _xmax && y < _ymax;}
    void next(unsigned& x, unsigned& y) const;
};
#endif


template <class CHAR>
terminal<CHAR>::terminal(CHAR initial_background)
: _background(initial_background),
_xmax((term_construct(), term_xmax())),
_ymax(term_ymax())
{
    if (_background != static_cast<CHAR>(' ')) {
        for (unsigned y = 0; y < _ymax; ++y) {
            for (unsigned x = 0; x < _xmax; ++x) {
                put(x, y);
            }
        }
    }
```

```
}

template <class CHAR>
inline terminal<CHAR>::~terminal()
{
    for (unsigned y = 0; y < _ymax; ++y) {
        for (unsigned x = 0; x < _xmax; ++x) {
            put(x, y, ' ');
        }
    }

    term_destruct();
}

template <class CHAR>
void terminal<CHAR>::put(unsigned x, unsigned y, CHAR c) const
{
    if (isprint(static_cast<CHAR>(c)) == 0) {
        ostringstream os;
        os << "unprintable character "
        << static_cast<unsigned>(static_cast<CHAR>(c))
        << ".\n";
        throw except(os);
    }

    check(x, y);
    term_put(x, y, c);
}


//Move to the next (x, y) position: left to right, top to bottom.
//Warning: will change the values of the arguments.
template <class CHAR>
void terminal<CHAR>::next(unsigned& x, unsigned& y) const
{
    check(x, y);

    if (++x >= _xmax) {
        x = 0;
        ++y;
    }
}

template <class CHAR>
void terminal<CHAR>::check(unsigned x, unsigned y) const
{
    if (!in_range(x, y)) {
        ostringstream os;
        os << "coordinates (" << x << ", " << y
        << ") must be >= (0, 0) and < ("
        << _xmax << ", " << _ymax << ")\n";
        throw except(os);
    }
}
```

```cpp
//main
#include <iostream>
#include <cstdlib>  //for the srand function and EXIT_SUCCESS
#include <ctime>    //for the time function
#include <new>      //for bad_alloc
#include <exception>    //for exception
#include "game.h"
#include "printable.h"
using namespace std;

int main(int argc, char **argv)
{
    int status = EXIT_FAILURE;  //guilty until proven innocent
    srand(static_cast<unsigned>(time(0)));

    try {
        game g = '!';
        g.play();
        status = EXIT_SUCCESS;
    }

    catch (const bad_alloc& bad) {
        cerr << argv[0] << ": new failed: " << bad.what() << "\n";
    }

    catch (const exception& e) {
        cerr << argv[0] << ": " << e.what() << "\n";
    }

    catch (...) {
        cerr << argv[0] << ": main caught unexpected exception.\n";
    }

    return status;
}
```

```
#ifndef GAMEH
#define GAMEH
#include <list>
#include "terminal.h"
#include "except.h"
#include "printable.h"
using namespace std;

class wabbit;   //forward declaration

class game {
    typedef terminal<printable> terminal_t;
    const terminal_t term;

    typedef list<wabbit *> master_t;
    master_t master;

    master_t::value_type get(unsigned x, unsigned y) const;
    master_t::size_type count(char c) const;
public:
    game(char initial_c = '.');
    ~game() {depopulate();}

    game(terminal_t::value_type initial_c = '.'):term(initial_c){}
    master_t::size_type count(terminal_t::value_type c) const;

    void play();
    void depopulate();
    friend class wabbit;
};
#endif
```

```
#ifndef WABBITH
#define WABBITH
#include "game.h"

class wabbit {
    game *const g;
    unsigned x, y;

    //move calls these functions to decide who eats who.  wabbit w1 will eat
    //wabbit w2 if w1.hungry() > w2.bitter(), i.e., if w1's hunger is
    //stronger than w2's bitterness.
    virtual int hungry() const = 0;
    virtual int bitter() const = 0;

    //move calls this function to decide which direction to move in.
    virtual void decide(int *dx, int *dy) const = 0;

    //move calls this function if this wabbit tries to move off the screen,
    //or bumps into another wabbit that it can neither eat nor be eaten by.
    //(Will also called by manual::decide.)
    virtual void punish() const {}

    wabbit(const wabbit& another);              //deliberately undefined
    wabbit& operator=(const wabbit& another);  //ditto

protected:
    char key() const {return g->term.key();}   //called by manual::decide
    void beep() const {g->term.beep();}        //called by manual::punish
    typedef game::terminal_t terminal_t;

private:
    const terminal_t::value_type c;
public:
    wabbit(game *initial_g, unsigned initial_x, unsigned initial_y,
           terminal_t::value_type initial_c);
    virtual ~wabbit();

    bool move();

    //Functions that use the private data members of class wabbit.
    friend game::master_t::value_type game::get(unsigned x,unsigned y)const;
    friend game::master_t::size_type game::count(char c) const;
};
#endif
```

```
#include <iostream>
#include <sstream>
#include "except.h"
#include "wabbit.h"
using namespace std;

/*
 Delete any other wabbit that got eaten during the move (line 32), but do not
 delete this wabbit.  If this wabbit was eaten during the move, return false
 (line 36); otherwise return true.
 */

wabbit::wabbit(game *initial_g, unsigned initial_x, unsigned initial_y,
               terminal_t::value_type initial_c)
: g(initial_g), x(initial_x), y(initial_y), c(initial_c)
{
    if (!g->term.in_range(x, y)) {
        ostringstream os;
        os << "Initial wabbit position (" << x << ", " << y
        << ") off " << g->term.xmax() << " by "
        << g->term.ymax() << " terminal.\n";
        throw except(os);
    }

    const char other = g->term.get(x, y);
    const char background = g->term.background();

    if (other != background) {
        ostringstream os;
        os << "Initial wabbit position (" << x << ", " << y
        << ") already occupied by '" << other << "'.\n";
        throw except(os);
    }

    if (c == background) {
        ostringstream os;
        os << "Wabbit character '" << c << "' can't be the same as "
        "the terminal's background character.\n";
        throw except(os);
    }

    g->term.put(x, y, c);
    g->master.push_back(this);
}

wabbit::~wabbit()
{
    g->master.remove(this);
    g->term.put(x, y);
}

bool wabbit::move()
{
    int dx;   //uninitialized variables
    int dy;
    decide(&dx, &dy);
```

```
    if (dx == 0 && dy == 0) {
        return true;
    }

    const unsigned newx = static_cast<int>(x) + dx;
    const unsigned newy = static_cast<int>(y) + dy;

    if (!g->term.in_range(newx, newy)) {
        punish();
        return true;
    }

    if (wabbit *const other = g->get(newx, newy)) {
        const bool I_ate_him =  this->hungry() > other->bitter();
        const bool he_ate_me = other->hungry() >  this->bitter();

        if (I_ate_him) {
            other->beep();
            other->g->term.wait(1000);
            delete other;
        }

        if (he_ate_me) {
            beep();
            g->term.wait(1000);
            return false;
        }

        if (!I_ate_him) {
            //I bumped into a wabbit that I could neither eat nor be
            //eaten by.
            punish();
            return true;
        }
    }

    g->term.put(x, y);      //Erase this wabbit from its old location.
    x = newx;
    y = newy;
    g->term.put(x, y, c);   //Redraw this wabbit at its new location.

    return true;
}
```