```
// City.cpp

#include "City.h"
#include "Player.h"
#include "Flatulan.h"
#include "History.h"
#include "globals.h"
#include <iostream>
#include <cstdlib>
using namespace std;

City::City(int nRows, int nCols)
 : m_rows(nRows), m_cols(nCols), m_player(nullptr), m_nFlatulans(0),
  m_history(nRows, nCols)
{
    if (nRows <= 0  ||  nCols <= 0  ||  nRows > MAXROWS  ||  nCols > MAXCOLS)
    {
        cout << "***** City created with invalid size " << nRows << " by "
             << nCols << "!" << endl;
        exit(1);
    }
}

City::~City()
{
    for (int k = 0; k < m_nFlatulans; k++)
        delete m_flatulans[k];
    delete m_player;
}

int City::rows() const
{
    return m_rows;
}

int City::cols() const
{
    return m_cols;
}

Player* City::player() const
{
    return m_player;
}

int City::flatulanCount() const
{
    return m_nFlatulans;
}

int City::nFlatulansAt(int r, int c) const
{
    int count = 0;
    for (int k = 0; k < m_nFlatulans; k++)
```

```
    {
        const Flatulan* fp = m_flatulans[k];
        if (fp->row() == r  &&  fp->col() == c)
            count++;
    }
    return count;
}

bool City::determineNewPosition(int& r, int& c, int dir) const
{
    switch (dir)
    {
      case UP:     if (r <= 1)      return false; else r--; break;
      case DOWN:   if (r >= rows()) return false; else r++; break;
      case LEFT:   if (c <= 1)      return false; else c--; break;
      case RIGHT:  if (c >= cols()) return false; else c++; break;
      default:     return false;
    }
    return true;
}

void City::display() const
{
      // Position (row,col) in the city coordinate system is represented in
      // the array element grid[row-1][col-1]
    char grid[MAXROWS][MAXCOLS];
    int r, c;

        // Fill the grid with dots
    for (r = 0; r < rows(); r++)
        for (c = 0; c < cols(); c++)
            grid[r][c] = '.';

        // Indicate each Flatulan's position
    for (int k = 0; k < m_nFlatulans; k++)
    {
        const Flatulan* fp = m_flatulans[k];
        char& gridChar = grid[fp->row()-1][fp->col()-1];
        switch (gridChar)
        {
          case '.':  gridChar = 'F'; break;
          case 'F':  gridChar = '2'; break;
          case '9':  break;
          default:   gridChar++; break;  // '2' through '8'
        }
    }

        // Indicate player's position
    if (m_player != nullptr)
    {
          // Set the char to '@', unless there's also a Flatulan there
          // (which should never happen), in which case set it to '*'.
        char& gridChar = grid[m_player->row()-1][m_player->col()-1];
        if (gridChar == '.')
```

```
            gridChar = '@';
        else
            gridChar = '*';
    }

        // Draw the grid
    clearScreen();
    for (r = 0; r < rows(); r++)
    {
        for (c = 0; c < cols(); c++)
            cout << grid[r][c];
        cout << endl;
    }
    cout << endl;

        // Write message, Flatulan, and player info
    cout << "There are " << m_nFlatulans << " unconverted Flatulans remaining." <<
     endl;
    if (m_player == nullptr)
        cout << "There is no player." << endl;
    else
    {
        if (m_player->age() > 0)
            cout << "The player has lasted " << m_player->age() << " steps." <<
             endl;
        if (m_player->isPassedOut())
            cout << "The player has passed out." << endl;
        else
            cout << "The player's health level is " << m_player->health() << endl;
    }
}

History& City::history()
{
    return m_history;
}

bool City::addFlatulan(int r, int c)
{
    if (! isInBounds(r, c))
        return false;

      // Don't add a Flatulan on a spot with a player
    if (m_player != nullptr  &&  m_player->row() == r  &&  m_player->col() == c)
        return false;

      // Dynamically allocate a new Flatulan and add it to the city
    if (m_nFlatulans == MAXFLATULANS)
        return false;
    m_flatulans[m_nFlatulans] = new Flatulan(this, r, c);
    m_nFlatulans++;
    return true;
}
```

```
bool City::addPlayer(int r, int c)
{
    if (! isInBounds(r, c))
        return false;

      // Don't add a player if one already exists
    if (m_player != nullptr)
        return false;

      // Don't add a player on a spot with a Flatulan
    if (nFlatulansAt(r, c) > 0)
        return false;

      // Dynamically allocate a new Player and add it to the city
    m_player = new Player(this, r, c);
    return true;
}

void City::preachToFlatulansAroundPlayer()
{
      // Preach to Flatulans orthogonally or diagonally adjacent to player.  If a
      // Flatulan is converted, then since the order of the Flatulans in the array
      // doesn't matter, we can replace the converted Flatulan we remove from the
      // game by the last one in the array.
    for (int k = 0; k < m_nFlatulans; )
    {
        Flatulan* fp = m_flatulans[k];
        int rowdiff = fp->row() - m_player->row();
        int coldiff = fp->col() - m_player->col();


        if (rowdiff < -1  ||  rowdiff > 1  ||
            coldiff < -1  ||  coldiff > 1)  // not orthogonally or diagonally
             adjacent
            k++;
        else if ( ! fp->possiblyGetConverted() )  // adjacent, but unconverted
    {
            m_history.record(fp->row(), fp->col());
            k++;
    }
        else  // converted
        {
            delete m_flatulans[k];
            m_flatulans[k] = m_flatulans[m_nFlatulans-1];
            m_nFlatulans--;
        }
    }
}

void City::moveFlatulans()
{
    for (int k = 0; k < m_nFlatulans; k++)
    {
        Flatulan* fp = m_flatulans[k];
```

```
        fp->move();
        int rowdiff = fp->row() - m_player->row();
        int coldiff = fp->col() - m_player->col();
          // if orthogonally adjacent
        if  ((rowdiff == 0  &&  (coldiff == 1  ||  coldiff == -1))  ||
             (coldiff == 0  &&  (rowdiff == 1  ||  rowdiff == -1)) )
            m_player->getGassed();
    }
}

bool City::isInBounds(int r, int c) const
{
    return (r >= 1  &&  r <= m_rows  &&  c >= 1  &&  c <= m_cols);
}
```