



## Programming Assignment 7

### It's a Gas!

**Time due: 11:00 PM Thursday, December 7**

You have arrived in the walled city of Flatula, whose residents eat beans, cabbage, and hard-boiled eggs with every meal. Although they are used to the resulting effects, the odor that emanates from the Flatulans you encounter makes you woozy. Your only hope to keep from passing out is to successfully preach to the Flatulans about converting to a diet of lean meat, lettuce, and rice.

That's the scenario for a new video game under development. Your assignment is to complete the prototype that uses character-based graphics.

If you execute [this Windows program](#) or [this Mac program](#) or [this Linux program](#), you will see the player (indicated by ) in a rectangular city filled with Flatulans (usually indicated by ). At each turn, the user will select an action for the player to take. The player will take the action, and then each Flatulan will move one step in a random direction. The attempt to move will make each Flatulan break wind. If the Flatulan is orthogonally adjacent to the player, the player is affected by the blast of gas. If the player suffers ten such blasts during the game, the player passes out and the game is over.

This smaller [Windows version](#) or [Mac version](#) or [Linux version](#) of the game may help you see the operation of the game more clearly.

At each turn the player may take one of these actions:

1. Move one step up, down, left, or right to an empty position (i.e., one not occupied by a Flatulan). If the player attempts to move into the wall of the city (e.g., down, when on the bottom row) or to a position occupied by a Flatulan, the player does not move.
2. Preach to the Flatulans adjacent to the player orthogonally or diagonally (i.e., to any Flatulans next to the player in the eight directions). For each of those Flatulans, there is a  $\frac{2}{3}$  probability that the player will convert that Flatulan to no longer pollute the air. A Flatulan who has been converted should be removed from the game, since they pose no further threat to the player and are thus irrelevant.

The game allows the user to select the player's action by typing u/d/l/r for moving or just hitting enter for preaching. The user may also type q to prematurely quit the game.

When it's the Flatulans' turn, each Flatulan picks a random direction (up, down, left, or right) with equal probability. The Flatulan moves one step in that direction if it can; however, if doing so would cause the Flatulan to move into a wall of the city (e.g., down, when on the bottom row) or to the position occupied by the player, it does not move. More than one Flatulan may occupy the same position; in that case, instead of `F`, the display will show a digit character indicating the number of Flatulans at that position (where 9 indicates 9 or more). After each Flatulan attempts to move (even if it doesn't actually move), if it is now orthogonally adjacent to the player (i.e., next to the player directly above, below, to the left, or to the right), the player suffers one blast of gas from that Flatulan. If this is the tenth gas blast the player has suffered during the game, the player passes out and the game is over.

Your assignment is to complete [this C++ program skeleton](#) to produce a program that implements the described behavior. (We've indicated where you have work to do by comments containing the text `TODO`; remove those comments as you finish each thing you have to do.) The program skeleton you are to flesh out defines four classes that represent the four kinds of objects this program works with: `Game`, `City`, `Flatulan`, and `Player`. Details of the interface to these classes are in the program skeleton, but here are the essential responsibilities of each class:

## Game

- To create a `Game`, you specify a number of rows and columns and the number of Flatulans to start with. The `Game` object creates an appropriately sized `City` and populates it with the `Player` and the Flatulans.
- A game may be played. (Notice that the city is displayed after the Flatulans have had their turn to move, but not after the player has had its turn.)

## City

- When an `City` object of a particular size is created, it has no Flatulans or player. In the `City` coordinate system, row 1, column 1 is the upper-left-most position that can be occupied by a Flatulan or `Player`. (If an `City` were created with 7 rows and 8 columns, then the lower-right-most position that could be occupied would be row 7, column 8.)

- You may tell the City object to create or a Flatulan at a particular position.
- You may tell the City object to create a Player at a particular position.
- You may tell the City object to have all the Flatulans in it make their move.
- You may tell the City object that the Flatulans around the player are being preached to. Flatulans that are converted must be removed from the City, since they serve no further purpose in the game.
- You may ask the City object its size, how many Flatulans are at a particular position, and how many Flatulans altogether are in the City.
- You may ask the City object whether moving from a particular position in a particular direction is possible (i.e., would not go off the edge of the city), and if so, what the resulting position would be.
- You may ask the City object for access to its player.
- An City object may be displayed on the screen, showing the locations of the Flatulans and player, along with other status information.

## Player

- A Player is created at some position (using the City coordinate system, where row 1, column 1 is the upper-left-most position, etc.).
- You may tell a Player to preach or to move in a particular direction.
- You may tell a Player it has suffered a gas blast.
- You may ask a Player for its position, age (i.e., how many turns the player has survived without passing out), and health status (i.e., how many more gas blasts the player will cause the player to pass out).

## Flatulan

- A Flatulan is created at some position (using the City coordinate system, where row 1, column 1 is the upper-left-most position, etc.).
- You may tell a Flatulan to move.
- You may ask a Flatulan object for its position.
- You may tell a Flatulan object that it is being preached to, and find out whether the Flatulan has been converted.

The skeleton program you are to complete has all of the class definitions and implementations in one source file, which is awkward. Since we haven't yet learned about separate compilation, we'll have to live with it.

Complete the implementation in accordance with the description of the game. You are allowed to make whatever changes you want to the *private* parts of the classes: You

may add or remove private data members or private member functions, or change their types. You must *not* make any deletions, additions, or changes to the *public* interface of any of these classes — we're depending on them staying the same so that we can test your programs. You can and will, of course, make changes to the *implementations* of public member functions, since the callers of the function wouldn't have to change any of the code they write to call the function. You must **not** declare any public data members, nor use any global variables whose values may change during execution (so global constants are OK). You may add additional functions that are not members of any class. The word `friend` must not appear in your program.

Any member functions you implement must never put an object into an invalid state, one that will cause a problem later on. (For example, bad things could come from placing a Flatulan outside the city.) Any function that has a reasonable way of indicating failure through its return value should do so. Constructors pose a special difficulty because they can't return a value. If a constructor can't do its job, we have it write an error message and exit the program with failure by calling `exit(1);`. (We haven't learned about throwing an exception to signal constructor failure.)

What you will turn in for this assignment is a zip file containing this one file and nothing more:

1. A text file named **gas.cpp** that contains the source code for the completed C++ program. This program must build successfully, and its correctness must not depend on undefined program behavior. Your program must not leak memory: Any object dynamically allocated during the execution of your program must be deleted (once only, of course) by the time your main routine returns normally.

Notice that you do not have to turn in a report describing the design of the program and your test cases.

By Wednesday, December 6, there will be a link on the class web page that will enable you to turn in your zip file electronically. Turn in the file by the due time above.