Fall 2021 CS 31

Programming Assignment 7
Gone with the Wind

Time due: 11:00 PM Thursday, December 2

Aliens have kidnapped you, and after feeding you a diet of beans, cabbage, and hard-boiled eggs for days, placed you atop a high mesa with steep drop-offs on all sides. For their bloodthirsty amusement, they also placed on the mesa hungry garks, fierce creatures who would happily devour you. The aliens informed you that garks are fatally vulnerable to foul odors and that your diet was designed to make it a fair fight by enabling you to be the source of those odors.

Well, 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 mesa filled with garks (usually indicated by `G`). At each turn, the user will select an action for the player to take. The player will take the action, and then each gark will move one step in a random direction. If a gark lands on the position occupied by the player, it devours the player and the game ends.

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. If the player attempts to move off of the mesa (e.g., down, when on the bottom row), the player does not move on this turn, since the game does not allow the player to leave the mesa.).
2. Attack an adjacent gark. If the player indicates up, down, left, or right, and there is a gark at that position, then this means the player does not move, but instead aims a release of gas at the gark. This startles the gark and weakens it; if this is the second time the gark has been attacked with gas, it dies. If this is only the first time the gark has been attacked, it instinctively steps back to the position behind it (i.e. adjacent to it on the opposite side from the side the player is on relative to the gark). If the gark is on the edge of the mesa, so there is no position behind it, then it falls off the mesa and dies. The position a gark steps back to may be occupied by other garks, since it is is allowable for multiple garks to occupy the same position. When the player attacks a position occupied by more than one gark, only one of those garks reacts to the gas by either dying or stepping back one position; the other(s) are unaffected (presumably because the affected gark happened to shield the other(s)).
3. Stand. In this case, the player does not move or attack.

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

When it's the garks' turn, each gark picks a random direction (up, down, left, or right) with equal probability. The gark moves one step in that direction if it can; if the gark attempts to move off of the mesa, however, (e.g., down, when on the bottom row), it does not move. More than one gark may occupy the same position; in that case, instead of `G`, the display will show a digit character indicating the number of garks at that position (where `9` indicates 9 or more). If after the garks move, a gark occupies the same position as the player, the player is devoured and dies.

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, Mesa, Gark, 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 garks to start with. The Game object creates an appropriately sized Mesa and populates it with a Player and the Garks.
- A game may be played. (Notice that the mesa is displayed after the garks have had their turn to move, but not after the player has moved. Therefore, if a player causes a gark to step back one position, the gark will have a chance to move before you see the display, so it might appear to have moved one more position back, or back and to the side, or, a glutton for punishment, right next to the player again.)

Mesa

- When a Mesa object of a particular size is created, it has no garks or player. In the Mesa coordinate system, row 1, column 1 is the upper-left-most position that can be occupied by a Gark or Player. (If a Mesa 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 a Mesa object to create or destroy a Gark at a particular position.
- You may tell a Mesa object to create a Player at a particular position.
- You may tell a Mesa object to have all the garks in it make their move.
- You may tell a Mesa object that a gark is being attacked, and find out whether the attack killed the gark.
- You may ask a Mesa object its size, how many garks are at a particular position, and how many garks altogether are in the Mesa.
- You may ask a Mesa object whether moving from a particular position in a particular direction is possible (i.e., would not go off the edge of the mesa), and if so, what the resulting position would be.
- You may ask a Mesa object for access to its player.
- A Mesa object may be displayed on the screen, showing the locations of the garks and the player, along with other status information.

Player

- A Player is created at some position (using the Mesa coordinate system, where row 1, column 1 is the upper-left-most position, etc.).
- You may tell a Player to stand or to move or attack in a direction. (The player ate a *lot* beans, so can launch an effectively unlimited number of attacks.)
- You may tell a Player it has died.
- You may ask a Player for its position, alive/dead status, and age. (The age is the count of how many turns the player has survived.)

Gark

- A Gark is created at some position (using the Mesa coordinate system, where row 1, column 1 is the upper-left-most position, etc.).
- You may tell a Gark to move.
- You may ask a Gark object for its position.
- You may tell a Gark object that it has been attacked, and find out whether that attack killed the gark.

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 gark outside the mesa.) 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 **garks.cpp** that contains the source code for the completed C++ program. This program must build successfully using both g31 and either Visual C++ or clang++, 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. For this project, you do not have to write comments in your program.

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

By Wednesday, December 1, there will be a link on the class webpage that will enable you to turn in your zip file electronically. Turn in the file by the due time above.