CS 31 Programming Assignment 7
Exterminator

Rats have overrun Pauley Pavilion! The pest control company you work for has supplied you with poison pellets and sent you in to exterminate the rats.

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 arena filled with rats (usually indicated by `R`). At each turn, the user will select an action for the player to take: either move one step or drop a poison pellet without moving. The player will take the action, and then each rat will move one step in a random direction. If a rat moves onto the grid point occupied by the player, the player dies, because the rats carry quick-acting toxic bacteria. (If the player moves to a grid point occupied by a rat, the player is bitten and dies, so that would be a dumb move.)

If a rat lands on a grid point with a poison pellet, it eats the pellet. The first time a rat eats a poison pellet, it slows down: it doesn't move on its next turn, but it moves on the turn after that, then on the next turn it doesn't move, then it moves on the turn after that, etc., moving only every other turn. The second time a rat eats a poison pellet, it dies.

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 north, east, south, or west, and do not drop a poison pellet. If the player attempts to move out of the arena (e.g., south, when on the bottom row), the player does not move, and does not drop a pellet. If the player moves to a grid point currently occupied by a rat, the player dies.
2. Do not move, but attempt to drop a poison pellet. If there is already a poison pellet at that grid point, no additional pellet is dropped; a grid point may have at most one poison pellet. The player has an unlimited supply of poison pellets.

The game allows the user to select the player's action: n/e/s/w for movement, or x for dropping a poison pellet. The user may also just hit enter to have the computer select the player's move.

After the player moves, it's the rats' turn. Each rat has an opportunity to move. A rat that has previously eaten a poison pellet will not move if it attempted to move on the previous turn. Otherwise, it will pick a random direction (north, east, south, west) with equal probability. The rat moves one step in that direction if it can; if the rat attempts to move off the grid, however, it does not move (but this still counts as a poisoned rat's attempt to move, so it won't move on the next turn). More than one rat may occupy the same grid point; in that case, instead of `R`, the display will show a digit character indicating the number of rats at that point (where `9` indicates 9 or more).

If after a rat moves, it occupies the same grid point as the player (whether or not there's a poison pellet at that point), the player dies. If the rat lands on a grid point with a poison pellet on it, it eats that pellet (so the pellet is no longer at that point). If this is the second poison pellet the rat has eaten, it dies. If more than one rat lands on a spot that started the turn with a poison pellet on it, only one of them eats the pellet.

The CS 31 assignment was to complete a C++ program skeleton to produce a program that implements the described behavior. For CS 32 Project 1, we will give you a correct solution to the CS 31 assignment. The program defines four classes that represent the four kinds of objects this program works with: Game, Arena, Rat, 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 rats to start with. The Game object creates an appropriately sized Arena and populates it with the Player and the Rats.
- A game may be played.

Arena

- When an Arena object of a particular size is created, it has no positions occupied by rats or the player. In the Arena coordinate system, row 1, column 1 is the upper-left-most position that can be occupied by a rat or the player. If an Arena created with 10 rows and 20 columns, then the lower-right-most position that could be occupied would be row 10, column 20.
- You may tell an Arena object to put a poison pellet at a particular position.
- You may ask an Arena object whether there's a poison pellet at a particular position.
- You may tell an Arena object to create a Rat at a particular position.
- You may tell an Arena object to create a Player at a particular position.
- You may tell an Arena object to have all the rats in it make their move.
- You may ask an Arena object its size, how many rats are at a particular position, and how many rats altogether are in the Arena.
- You may ask an Arena object for access to its player.
- An Arena object may be displayed on the screen, showing the locations of the rats, the player, and the poison pellets, along with other status information.

Player

- A Player is created at some position (using the Arena coordinate system, where row 1, column 1 is the upper-left-most position, etc.).
- You may tell a Player to move in a direction or to drop a poison pellet.
- You may tell a Player that it has died.
- You may ask a Player for its position and its alive/dead status.

Rat

- A Rat is created at some position.
- You may tell a Rat to move.
- You may ask a Rat object for its position and its alive/dead status.

CS 31 students were told:

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, 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 rat outside the arena.) If a function has a reasonable way of indicating failure through its return value, it 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.)