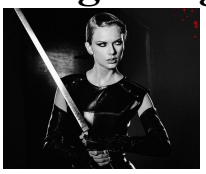
Fall 2019 CS 31

Programming Assignment 7



Bad Blood

Time due: 11:00 PM Thursday, December 5

Vampires have overrun Pauley Pavilion! The pest control company you work for has supplied you with vials of poisoned blood and sent you in to exterminate the vampires.

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 vampires (usually indicated by `V`). At each turn, the user will select an action for the player to take: either move one step or drop a poisoned blood vial without moving. The player will take the action, and then each vampire will move one step in a random direction. If a vampire moves onto the grid point occupied by the player, the player dies from the vampire's bite. (If the player moves to a grid point occupied by a vampire, the player is bitten and dies, so that would be a dumb move.)

If a vampire lands on a grid point with a poisoned blood vial, it drinks all the blood in the vial. The first time a vampire drinks a vial of poisoned blood, 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 vampire drinks a vial of poisoned blood, it dies and turns to dust. (If a poisoned vampire moves to a grid point with both the player and a vial of poisoned blood, it drinks the blood in the vial and just before it dies, it bites the player, so both die.)

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 poisoned blood vial. 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 vial. If the player moves to a grid point currently occupied by a vampire, the player dies.
2. Do not move, but attempt to drop a vial of poisoned blood. If there is already a vial of poisoned blood at that grid point, no additional vial is dropped; a grid point may have at most one vial of poisoned blood. The player has an unlimited supply of poisoned blood vials.

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

After the player moves, it's the vampires' turn. Each vampire has an opportunity to move. A vampire that has previously drunk a vial of poisoned blood 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 vampire moves one step in that direction if it can; if the vampire attempts to move off the grid, however, it does not move (but this still counts as a poisoned vampire's attempt to move, so it won't move on the next turn). More than one vampire may occupy the same grid point; in that case, instead of `V`, the display will show a digit character indicating the number of vampires at that point (where `9` indicates 9 or more).

If after a vampire moves, it occupies the same grid point as the player (whether or not there's a vial of poisoned blood at that point), the player dies. If the vampire lands on a grid point with a poisoned blood vial on it, it drinks all the blood in the vial (and the vial is no longer part of the game). If this is the second vial of poisoned blood the vampire has drunk, it dies. If more than one vampire lands on a spot that started the turn with a poisoned blood vial on it, only one of them drinks the vial of poisoned blood.

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, Arena, Vampire, 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 vampires to start with. The Game object creates an appropriately sized Arena and populates it with the Player and the Vampires.
- A game may be played.

Arena

- When an Arena object of a particular size is created, it has no positions occupied by vampires or the player. In the Arena coordinate system, row 1, column 1 is the upper-left-most position that can be occupied by a vampire 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 poisoned blood vial at a particular position.
- You may ask an Arena object whether there's a poisoned blood vial at a particular position.
- You may tell an Arena object to create a Vampire 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 vampires in it make their move.
- You may ask an Arena object its size, how many vampires are at a particular position, and how many vampires 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 vampires, the player, and the poisoned blood vials, 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 poisoned blood vial.
- You may tell a Player that it has died.
- You may ask a Player for its position and its alive/dead status.

Vampire

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

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, 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 vampire 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.)

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

1. A text file named **vampires.cpp** that contains the source code for the completed C++ program. This program must build successfully under both g++ with Linux 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.

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

By Wednesday, December 4, there will be 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.