CS 31 Programming Assignment 7
Club Zombie

Exploring the city one evening, you enter an apparent nightspot intriguingly named Club Zombie. You're immediately grabbed, armed with a club, and dropped into an arena where you're pitted in combat against zombies for the amusement of the bloodthirsty patrons. 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 zombies (usually indicated by `Z`). At each turn, the user will select an action for the player to take. The player will take the action, and then each zombie will move one step in a random direction. If a zombie lands on the position occupied by the player, it eats the player's brain and the player 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 up, down, left, or right to an empty position. If the player attempts to move out of the arena (e.g., down, when on the bottom row), the result is the same as standing (i.e., not moving at all).
2. Attack an adjacent zombie. If the player indicates up, down, left, or right, and there is a zombie at that position, then this means the player does not move, but instead clubs the zombie. This injures the zombie; if this is the second time the zombie has been injured, it is destroyed. If this is only the first time the zombie has been injured, the momentum from the club swing knocks the zombie 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 zombie). If the zombie is on the edge of the arena, so there is no position behind it, then it is destroyed (presumably because being knocked into the wall of the arena deals it a second injury, enough to destroy it). A zombie knocked back to a position occupied by one or more zombies (it is allowable for multiple zombies to occupy the same position) does not suffer additional injury and does not injure those zombies or knock them to another position. When the player attacks a position occupied by more than one zombie, only one of those zombies is injured and either destroyed or knocked back one position; the others are unaffected.
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 zombies' turn, each zombie picks a random direction (up, down, left, or right) with equal probability. The zombie moves one step in that direction if it can; if the zombie attempts to move out of the arena, however, (e.g., down, when on the bottom row), it does not move. More than one zombie may occupy the same position; in that case, instead of `Z`, the display will show a digit character indicating the number of zombies at that position (where `9` indicates 9 or more). If after the zombies move, a zombie occupies the same position as the player, the player dies.

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, Zombie, and Player. Details of the interface to these classes are in the program, 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 zombies to start with. The Game object creates an appropriately sized Arena and populates it with the Player and the Zombies.
- A game may be played. (Notice that the arena is displayed after the zombies have had their turn to move, but not after the player has moved. Therefore, if a player knocks a zombie back one position, the zombie will have a chance to move before you see the display, so it might appear to have moved one more position back, or to the side, or, a glutton for punishment, back next to the player.)

Arena

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

Zombie

- A zombie 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 Zombie to move.
- You may ask a Zombie object for its position.
- You may tell a Zombie object that it has been attacked, and find out whether that attack destroyed the zombie.

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 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 zombie outside the arena.) 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.)