Fall 2015 CS 31

Programming Assignment 7
Rage Against the Machines

Time due: 9:00 PM Thursday, December 3

The Mechanoids of Planet Zork have captured you! To satisfy their lust for oilshed and bloodshed, they have armed you with a club and pitted you in combat against killer robots. 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 killer robots (usually indicated by `R`). At each turn, the user will select an action for the player to take. The player will take the action, and then each robot will move one step in a random direction. If a robot lands on the position occupied by the player, 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 robot. If the player indicates up, down, left, or right, and there is a robot at that position, then this means the player does not move, but instead clubs the robot. This damages the robot; if this is the second time the robot has been damaged, it is destroyed. If this is only the first time the robot has been damaged, the momentum from the club swing knocks the robot 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 robot). If the robot 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 does a second amount of damage, enough to destroy it). A robot knocked back to a position occupied by one or more robots (it is allowable for multiple robots to occupy the same position) does not suffer additional damage and does not damage those robots or knock them to another position. When the player attacks a position occupied by more than one robot,

only one of those robots is damaged 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 robots' turn, each robot picks a random direction (up, down, left, or right) with equal probability. The robot moves one step in that direction if it can; if the robot attempts to move out of the arena, however, (e.g., down, when on the bottom row), it does not move. More than one robot may occupy the same position; in that case, instead of `R`, the display will show a digit character indicating the number of robots at that position (where `9` indicates 9 or more). If after the robots move, a robot occupies the same position as the player, the player 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, Arena, Robot, 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 robots to start with. The Game object creates an appropriately sized Arena and populates it with the Player and the Robots.
- A game may be played. (Notice that the arena is displayed after the robots have had their turn to move, but not after the player has moved. Therefore, if a player knocks a robot back one position, the robot 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 robots or player. In the Arena coordinate system, row 1, column 1 is the upper-left-most position that can be occupied by a Robot 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 Robot 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 robots in it make their move.
- You may tell the Arena object that a robot is being attacked, and find out whether the attack destroyed the robot.
- You may ask the Arena object its size, how many robots are at a particular position, and how many robots 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 robots 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.)

Robot

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

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 robot 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.)

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

1. A text file named **robots.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 2, 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.