1 Introduction

This project is an extension of Project 9. In this project, you’ll be adding features to the Robots game which make it more playable. Key amongst these new features are teleports and a level system.

Along the way, we’ll be making some structural upgrades to the design of your program. You’ll be using a constructor, and private variables; you will also be adding the ability to support multiple board sizes.

Throughout this spec, you should assume that all of the requirements from Project 9 still apply - unless I specifically say otherwise.
2 The New Gameplay Elements

2.1 Teleport
In Project 10, we’ll be implementing the ability to “teleport” - that is, to send the player to random places on the board. Just like any other move, the robots all move after the teleport.

The player can teleport as often as they like - however, there are a limited number of “safe” teleports. A safe teleport is one that is guaranteed not to drop you next to (or on top of!) a robot or pile of rubble. That is, a safe teleport is guaranteed not to kill you. Once a player’s safe teleports are gone, they can still teleport - but the location will be chosen randomly, and often the player will end up next to a robot (or directly on a pile of rubble). When the robot moves, it will kill the player; in the case of rubble, the player is killed simply by landing on it.

2.2 Shovable Rubble
In Project 10, it’s impossible to walk onto any rubble. Instead, if a player tries to move into a square where there is rubble, he tries to “shove” it ahead. If the rubble has an empty space behind it (or a robot!), then this works; the rubble is moved, and the player moves to where the rubble used to be. If not (that is, the rubble has more rubble behind it, or it would have gone off of the board), then nothing happens.

2.3 Levels
Project 10 also adds levels of difficulty. When you clear the board of robots, the game will reset - but with more robots this time! You keep playing until you are killed by the robots; normally, this happens when you run out of safe teleports, and eventually some teleport takes you right next to a robot.

2.4 Status Text
In addition, the board is (slightly) updated: It includes (on the top, above the board) text which shows the current level, the number of robots remaining on the board, and the number of safe teleports left.

2.5 Resizable Board
In this program, the size of the board is given as a command-line parameter; if no parameter is given, then it defaults to 21.
3 New main() Method

I’ve provided an updated version of the main() method. I strongly encourage you to read through this method first - especially the part where it handles the teleport mechanism. It will be a lot easier to understand what your Proj10GameState class is supposed to do if you understand main() first.

Download the class Proj10Robots from http://www.cs.arizona.edu/classes/cs127a/spring16/projects/proj10_code.

You must not change this file. If you turn in an alternate version of the file, we’ll ignore it. This means that you must design the class Proj10GameState so that it works well with this class.

4 class Proj10GameState

You must implement the Proj10GameState class. Refer to the Project 9 spec to understand most of what it does; this section will only list the changes from Project 9.

4.1 Class Design Requirements

In Project 10, you’ll practice with using private and public, amongst other things. Your code must fulfill the following rules:

- Every instance variable must be private
- static variables are not allowed.
- None of the instance variables may be initialized where they are declared; instead, each must be initialized only in the class constructor.
- Every method called by main() must be public. (If you don’t obey this, your code won’t even compile!)

4.2 Constructor Instead of init()

In Project 9, there was an init() method. This method must be removed; use a constructor instead. The constructor must take a single parameter (an int), which gives the size of the board. That is, if the parameter is 20, then the board must by 20x20.

The constructor must perform the following tasks:

- Throw an IllegalArgumentException if the requested size is less than 5.
- Initialize all instance variables, including the array to store the board.
  - The player must be in the center of the board. If the board has an odd size, then the player must start in the exact middle; if the board has an even size, then rounding down is OK.
- The board must not have any robots or rubble.

- Call `StdDraw.setXscale()` and `StdDraw.setYscale()` to set the proper scale. (These methods must \textbf{not} be called anywhere else in the program.) These methods work exactly like `StdDraw.setScale()`, except that they set the scales for the X and the Y axes independently. The Y scale must use a slightly larger scale, in order to leave space for the status text at the top of the window.

- Call `StdDraw.show(0)` to turn on double-buffering mode.

Except for setting the scale and turning on double-buffering mode, the constructor must not do any drawing.

### 4.3 New Method: \texttt{allRobotsDestroyed()}

This method, like \texttt{isGameOver()}, returns a \texttt{boolean}; it indicates whether all of the robots have been destroyed.

### 4.4 Updates to \texttt{handleKeyTyped()}

This method must be updated to handle the rubble-shoving logic. Shoving happens when the player is adjacent to a pile of rubble and attempts to move onto it. Instead of moving onto it, the player “shoves” it ahead.

The rule is this: try moving the rubble one space, in the same direction as the player is moving. (If the player is moving diagonally, then move the rubble in the same diagonal direction.) If the space behind the rubble is either empty, or if it contains a robot, then the rubble can be shoved. If the space is already full of rubble - or if it is off of the board - then the rubble cannot be moved (and so the player cannot move, either).

If the rubble can be shoved, then both the rubble and the player move; if it cannot, then neither of them move.

#### 4.4.1 Teleporting

You don’t need to handle the teleport logic (other than implementing \texttt{dup()} and \texttt{doTeleport()} below). \texttt{main()} will notice when the player hits the spacebar or number 0, and will handle those; those characters will \textbf{not} be sent to this method.

### 4.5 New Method: \texttt{dup()}

This method must return a \texttt{new Proj10GameState} object. It allocates the new object, copies over all of the variables - including creating a \texttt{duplicate} of the board.

It is \textbf{critical} that you duplicate the board in this method, because this method is used by \texttt{main()} to explore possible teleport locations. Very often,
main() may explore some possibility, and then discard it - and if so, main() will expect that the original game state has not been modified!

4.6 New Method: doTeleport()

This method should teleport the player to a new, randomly selected location. This method should not do any checking to see if the location is safe; it’s even OK if the player lands directly on top of rubble or a robot (either of which would kill them). main() will do all of the work necessary to find a safe teleport location - by duplicating the game, and then randomly teleporting.

4.7 Updates to draw()

The logic of draw() is mostly unchanged, but it now has two parameters, both int: the current level, and the current number of safe teleports.

In addition to drawing the board, draw() must also print the status message across the top: it must tell the player the current level, the number of robots left on the board, and the number of remaining safe teleports.

Use StdDraw.text(double, double, String) to do this. You may implement this with a single call (with one long String, or with several different calls.

5 Pseudocode Requirements

No pseudocode for this, the last project!

6 Program Requirements

To receive most (but not all) of the points for this assignment, your program must:

- Create a class named Proj10GameState. It should support all of the same requirements as Proj09GameState, except as explicitly listed above.
- Implement all of the new requirements listed above.
- Support the keys w,a,s,d,x as detailed in the description of the handleKeyTyped() method.

To receive full credit, your program must additionally do the following (the same as Project 9):

- Implement diagonal moves, using the keys q,e,z,c.
  Note that the diagonal directions must support rubble-shoving logic, just like the rest.
• Support capital versions of all keys, as well as the numeric keypad (1-9). (That is, '7' should do the same thing as 'q', and so on.)
• Improve the picture of the robots, rubble, and player. Each picture must include at least three drawing elements (squares, rectangles, lines, etc.).

7 Java code

Your class must have exactly the name given above: Proj10GameState. Make sure that your Java file is named to match.

7.1 Header Comment

Every Java file that you submit must have a Header Comment - which is a simple comment, at the top of your file, which gives basic information about it. A header comment must include:

• The name of the Java class
• The name of this class (“CSc 127A Spring 16”)
• The assignment name
• Your name
• Your Section Leader’s name (or letter)
• A quick description of the Java class.

Here’s my suggested header comment for Program 10:

/* class Proj10GameState
 * CSc 127A Spring 16, Project 10
 * Author: TODO: your name here
 * SL Name: TODO: the name of your SL here
 * ---
 * TODO: put a quick (2-3 sentence) description of the class here */

Late Day Clarifications

Java code

Starting with Project 3, if you email us your code (instead of using turnin), it will count as a Late Day, even if you do it before the deadline. Please use
turnin. And remember: if you are having trouble connecting from home, it’s always possible to come to the 228 lab, and upload your file using a thumbdrive.

8 Turning in Your Programs

See the class website for information about how to upload your files to lectura, and how to use the ‘turnin’ utility.

8.1 Java Code

You must turn exactly one file, named exactly this:

Proj10GameState.java

You must turn in the Java file(s) using the assignment name

cs127a_s16_sX_proj10

(replace the X with your section letter).