Space Rover Project 0

February 20, 2017

Please don't be intimidated by the length of the project description. Even thought this project may sound longer than anything you have done so far, you already know everything that is necessary to implement it. Begin by reading this description carefully several times, and understanding specific details of the project.

Stranded on a lonely planet

The year is 2022, personal space transport is common now and you have taken your space ship out exploring. You found an ancient abandoned planet and decided to land and investigate.

There was a huge storm as you came down and your space ship now lies in pieces on a landing pad. There is no oxygen but you are able to move around in your rover. You have a repair manual which tells you how to fix your ship but you don't have the needed spare parts.

You can see some strange items scrattered around on the ground. Maybe they will be useful. You can also see some strange glowing orbs big enough to swallow your rover. When you touch one, you get sucked in and and appear in another place hovering above another of the glowing orbs. The orb was a portal!! When you re-enter the same portal, it takes you back where you came from.

Your goal is to move about the portal system and collect enough spare parts to fix your ship.

The Rover Game

Your are going to develop a computer program to run the Rover game.

The screen has several areas

- Your current task to fix the ship, and the parts you'll need for that task.
- Your inventory.
- The board showing you and your surroundings. These may also include items, the ship, and portals.
- Navigation buttons.
- Action buttons (pick up, return to ship, do task, quit, help).

Eclipse

You will be using Eclipse for the project. When you need to create text files (not Java classes), you can do so by selecting `File` → `New` → `File`, clicking on the destination folder, and entering a filename. Test plans, and user manuals will be best created this way.

Version control

As you work on the project, remember to frequently save your work, and keep several copies of it under different names. Remember to keep a copy of your most recent work on the cloud storage (in your email, Google Drive, or Dropbox). This way even if you lose your computer, you will be able to recover the project code.

GUI, Interfaces

A Graphical User Interface (GUI - pronounced *gooey*) is a modern way to interact with software. It is the frame and any buttons or menus you see in software such as Firefox, Eclipse, Word, etc. GUIs also contain the ability to display information in chart or pictorial format.

You are provided with a GUI for the Rover game. The GUI displays the task, the inventory, the room, and buttons.

You must use the provided GUI in your project. In order to use the GUI, you must call its methods and provide it with certain methods and arrays. These requirements are specified in an interface.

- **Gui.java** This is the top-level class in the Rover implementation. This class implements the actual GUI. You can see its public methods in the provided Javadoc. You do not need to edit this class.
- **GuiPanel.java** This class implements the portion of the GUI which draws the game board. You can see its public methods in the provided Javadoc. You do not need to edit this class.

  The `Gui` has a `GuiPanel` instance field (member data).
- **IRover.java** This interface defines all of the methods that the Rover will need. The `Gui` and `GuiPanel` both have an `IRover` instance field (member data).
- **IRoom.java** This interface defines all of the methods that the Room will need. The `GuiPanel` uses an `IRoom` local variable.
- **Rover.java** This class implements the `IRover` interface and provides most of the functionality for the game. There is only one instance of the `Rover` in the game. A `Rover` has instance fields for the current `Room`, the task queue, the inventory list, and the map back to base (explained below). The `Gui` and `GuiPanel` will call the methods in the interface. You implement this class.

- **Room.java** The class implements a room on the game board, it is up to you how to implement it. I would suggest a two dimensional array – but the final choice is yours. The `GuiPanel` will call the methods in the interface. The room needs a SIZE public static instance field. The size must be 16 in order to fit in the window.You implement this class.
- All other classes are up to you!

The main method for this project is located it the Gui. You are allowed to edit the Gui but it is not required to do this assignmnent and I suggest you get it working before you tamper with the Gui.

Things in the game

Your rover will see parts of the broken ship in a clump in the middle of the first room. The parts must be adjacent (touching one another). You can decide what goes where.

- The ship must have at least two different kinds of parts and at least 4 parts total.
- At least 3 of the ship parts must be broken.
- There will be a random assortment of stuff scattered around the room. Stuff might include cake, screws, gears, etc.. You choose which items and how many. Randomness makes the game cooler.
- There will be at least 2 portals.
- Rooms after the first one will have stuff and portals but not any ships. (Unless you like to decorate with discarded wrecks..up to you).

The Rover

The rover wanders around the room. It can go everywhere (it hovers over the ship parts).

- The rover can pick up the stuff (cake, gears, etc..) lying around the room.
- The rover cannot pickup the portals or the ship.
- The rover will use the direction buttons or the arrows on the keyboard to move around. Movement or pickup will cause the `Gui` to call the corresponding `IRover` methods. After each call, the Gui is automatically redrawn when the `GuiPanel` calls IRover.getRoom().
- The rover must move (if possible) when the directional buttons or keys are pressed.
- The rover must not fall off the edges of the room. Do not throw any uncaught exceptions either.
- If the rover is standing on an item and trys to pick it up it succeeds.

- If the rover is standing on empty space, a portal, or a ship and tries to pick it up, nothing happens.
- If the rover steps on a portal, it is transported to another room. The `GuiPanel` will automatically redraw the room when it calls IRover.getRoom().

Portals

Portals are interdimentional doorways to other rooms. If the rover steps on a portal, it must go through the portal. It will arrive on a portal in another room (and not get sucked back through till it steps off and on again). You can connect the portals how you like. Portals must be connected in a permanent 1 to 1 relationship. (If portal A connects to portal B, no other portal connects to A or B and the portals are never disconnected).

I suggest you consider generating random rooms if the rover steps on a new portal but that is up to you. The universe must be large enough that the rover can find tons of items to fix the ship.

If you decide to generate a specific number of rooms, you must generate at least 10 rooms.

Images

Several images have been provided for you. Feel free to find better ones. The rover comes with one image. It should only need that one image. The portal comes with two images; one for normal usage and one for showing the way back to the ship. The items each come with one image. (cake, gear, screw, ...). You can add any new items and change the images as you like. The ship parts each come with a working image and a broken image. We have provided cabin, engine, ramp and exhaust images. You can add any new ship parts and change the images as you like. The images should all be in the outer folder of your project (not in the src folder).

To create an image for this game, find an image. (document where you got it in a comment!) The image can be jpg or gif (other formats might be ok too). In a graphic editor (google for a free one), resize the image to be 25x25. Put this image with the others and you can use it.

Images in the Rover game are using the BufferedImage class. Here is how to read in an image.

```
BufferedImage image;

try {
    image = ImageIO.read(new File("rover.jpg"));
} catch (IOException e) {
    image = null;
}
```

ADTs

You will be writing and/or using three ADTs in this assignment; a list, a stack, and a queue. You must write one of them from scratch using linked nodes or an array. You must use the provided class from Java for another. You can do what you want for the third.

When using a provided class, you can use it directly or keep it as an instance field or inherit it as a subclass. For example, if you use the provided Java class for the task queue, you could choose to use Java's provided Stack class

```
Stack<Task> myTasks = new Stack();
```

or you could write a stack class which uses Java's Stack or List

```
public class MyStack {
  List<Task> tasks = new List();
}

MyStack tasks = new MyStack();
```

or you could choose to inherit Java's Stack or List.

```
public class MyStack extends Stack { }

MyStack tasks = new MyStack();
```

If you think of any further options, check with the instructor to make sure they are allowed.

Big-O Analysis

You should do a Big-O Analysis of every method in the ADT you implement from scratch. If you implement more than one ADT from scratch, then only analyze **one** of them. You will be graded on accuracy of your analysis. If you analyze performance for collection types, denote the size of the collection by n. Give your answer in the top comment of each method. You do not have to show all work inside the method.

Tasks

Tasks are shows as strings in the task field of the `Gui`. You can implement tasks however you like (using good design) but they must adhere to the following:

- Tasks are stored in a **queue**. The top task in the queue is the one that is displayed in the Gui.
- Each task is dispalyed with a name.
- Each task must have exactly 3 types of supplies that it uses.
- It must use at least one of each of those types of supplies.

- The string returned by IRover.getTask() must be formatted to use at most 4 lines. Here is an example:

  ```
  Fix the engine
  3 Screws
  2 Cakes
  1 Cabbage
  ```

  The exact wording or contents of those lines is up to you.

When the rover has picked up enough items to complete the task, it can go back to the ship and hover over the relevant broken portion. Then the player hits the "Perform Task" button. If the rover actually has enough of the correct items in its inventory, the items get used up, the task is completed, that ship part is now fixed, and the task queue dequeues the task. The `Gui` will display the next task when it calls IRover.getTask() again.

Be flexible. If you have 3 broken ramps and the task says "Fix the ramp", it need to work on any broken ramp. If it says "Fix the leftmost ramp", it should only work on the leftmost ramp.

If there are no further tasks, IRover.getTask() should return some sort of victory message instead of another task.

Inventory

The inventory is a list of parts the rover is carrying. The inventory shows as a string in the inventory field of the `Gui`. You can implement it however you like (using good design) but it must adhere to the following:

- Items in the inventory must be stored in a **List**.
- Items in the inventory must have names (such as Cake or Cabbage).
- Items must be displayed with their count (3 if the rover has 3 cabbages).
- Items must not be displayed if the rover does not have any.
- The string returned by IRover.getInventory() must fit neatly in the space provided for the inventory.

For example, your inventory might look like:

```
3 Cabbages
1 Screw
2 Gears
```

Map back to the ship

As your robot travels through the portals, you must use a **stack** to keep track of the way back home.

- When the player hits the "Show the way back button", the `Gui` will call IRover.showTheWayBack() and you must change the image for the portal on top of the stack. Do not go charging towards home. Let the player walk.
- As soon as the rover goes through any portals in that room, revert all portal images to normal.
- When the rover goes through the portal on top of the stack, pop the stack.
- When the rover goes through another portal, push it onto the stack.

Your stack can contain any type you want. I suggest a stack of portals or rooms.

Technical Specification

A technical specification explains all of the functionality of software and how it is assembled. **Complete your technical specification before you implement the code inside your methods!!!**

UML

Create UML diagram that includes all of the public and private methods as well as any member data, and class relationships.

Javadoc

This technical specification will show the complete design work for your software. It is only intended to be read by programmers and managers (not the user).

Before implementing the program, write a technical specification for the software including

1. A problem statement describing the assignment in your own words; put this problem statement at the top of your `Rover` class in a Javadoc comment. What does the software do from a user's point of view? (2 players, surrounded markers, etc ...)
2. A description of what the program needs to do; put this description in your `Rover` class under the problem statement. How does this software accomplish its task? (arrays, recursion, etc ...)
3. A Javadoc comment explaining the purpose of each other class you write; put this comment at the top of the class.
4. A Javadoc comment on each method that you write. This comment will stating its purpose, inputs, outputs, any methods it calls, and any (high level) algorithms it uses to accomplish the task. These algorithm would include the list of base cases and recursive cases for the board update.
5. A Javadoc comment on all instance fields in your classes. The comment should state what the field is used for.

Test plan

A test plan is a set of test cases created before you implement the software that you can run to see if your implementation is correct.

The test cases must test out movement, walls, picking things up, not picking up portals or ships or empty spaces, going through unknown portals, going back through known portals (the one you came in through), performing tasks that succeed, performing tasks that fail, and winning the game by running out of tasks.

Some tasks can be tested with automated testing, and you should write JUnit tests for them. Some tasks can be hard to automate with JUnit (e.g. testing GUI) and you should describe your tests as text scenarios.

Advanced Java

This section covers features of Java you may not have used before.

Casting types from a superclass to a subclass

If you use inheritance, you may have several child classes of one parent class. If you have a method which returns an object of type Parent and you know it is really the child type, you can cast it to the child type.

```
public Parent getParentItem();

Child c = (Child) getParentItem();
```

Finding the type of an object

If you use inheritance, you may have several child classes of one parent class. You can find out if an objct is a certain type.

```
void method(Parent parent) {
    if (parent instanceof Child)
        c = (Child) parent;
    ...
}
```

Using the iterator for-each loop

If you have an object which is iteratable (array, Java's linked list, etc..) you can go through its contents using a special for loop.

```
LinkedList<Integer> list = new LinkedList();
// add lots of ints to the list

// sum up the list
int sum = 0;
```

```
for(Integer i : list)
    sum = sum + i;
```

Warning: do not attempt to add or remove items from the list while in the for-each loop, This will cause concurrency exceptions to be thrown.

When using a for-each loop on a two-dimentional array, you need a for-each for both dimentions (just like normal for loops).

```
int[][] array = new int[20][30];
// fill the array with ints

// sum up the array
int sum = 0;
for(int[] row : array)
    for(int i : row)
        sum = sum + i;
```

Other Java features

The implementation of this project is up to you. You may want to read up on Java generics, static inner classes, the equals and compareTo and toString methods, abstract classes, static or final instance fields, try and catch blocks, enums, interfaces, inheritance, multi dimentional arrays, random numbers, and switch statements.

User Manual

Your user manual describes how the program should be used and what are the valid inputs from a user. They tell the user what Operating Systems it runs on, how to launch and quit the software, and what buttons to hit to run it. Assume your user can be told to run Eclipse but does not know Java so they are just clicking on the buttons you tell them to hit.

Unless you find a nicer way to accomplish this, it can be a text file. (Yes, you can use doc, docx, odf, rtf, or pdf instead of txt if you want to use a fancier editor).

Extra Credit Options

Here are some extra credit options.

- Items in the inventory are displayed in alphabetical order.
- Give the rover a weight limit and refuse to pick up items that put it over the limit. Display the current weight and limit in the inventory.
- Is it possible for your stack to contain travel loops such as portals A, B, C, B, C, D? Reason whether your implementation has this inefficiency. (Your reasoning must match your implementation).
- Have the help button display your user manual.
- Come up with a simple but nifty use for an ADT in this project, explain it, and implement it.

Code Re-use

You may use any code that you worked with in the labs and in class, you may use any code found in the zybook or on any of the assigned readings. You may not get code from other people or use code from other parts of the internet or other books. (And the readings and labs already contain practically all the code you'd ever want).

Flow

There is only one phase for this project. I suggest you remember what you learned about software design and do your design (UML, Javadoc, and test plan) first. See your instructor and TAs if you need help along the way.

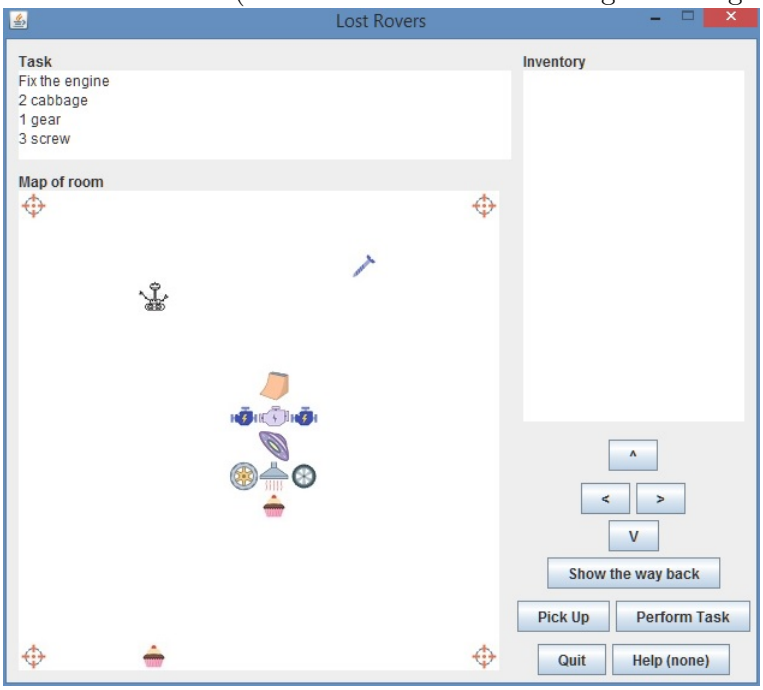
Handin

- Javadoc for all files
- UML
- Your code (commented)
- Game must play as described above.
- One ADT must be implemented from scratch.
- One ADT must use Java's ADTs.
- An ADT written from scratch must have a complete Big-O analysis.
- Testplan
- User Manual

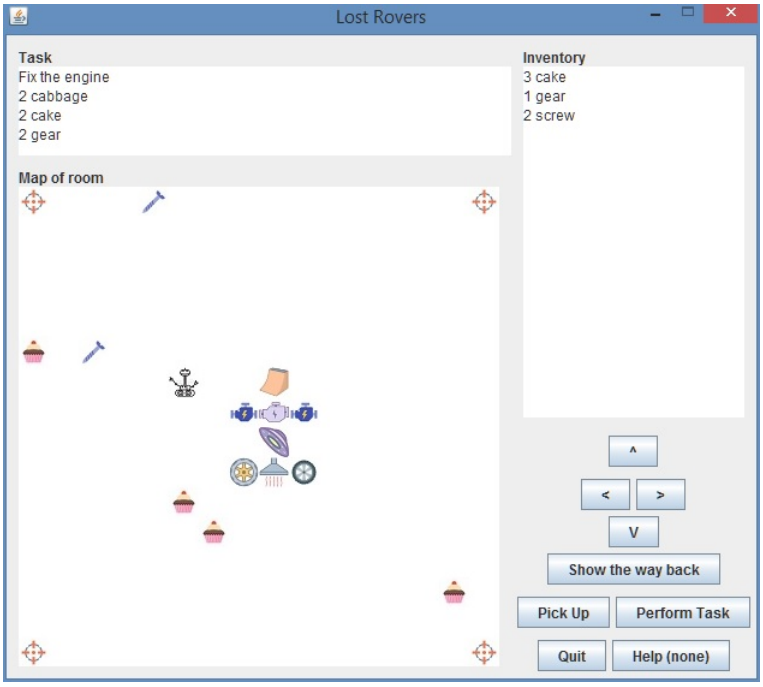
Extra Credit: If you did any extra credit on this assignment, make sure to note it at the top of the Rover class. Its possible you might need to explain it in your User Manual. You definitly need comments explaining it.

Screenshots

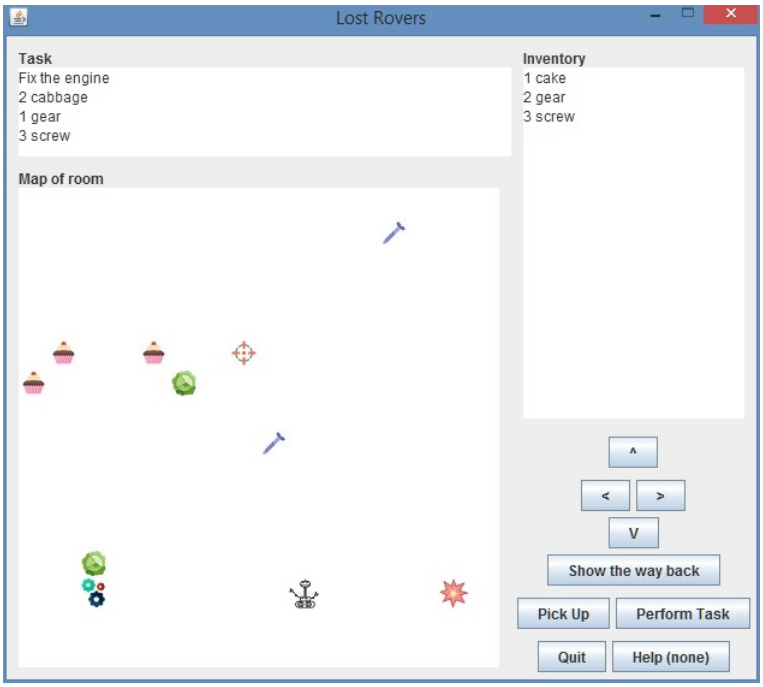
At the start of the game, we see a broken ship in the center. Three blue parts on one row are engines, and one engine part is broken. The UFO purple thing is the cabin, below it is the exhaust pipe. The yellow thing on top of engine is the ram. The target-like objects are the portals. The cakes and screws are hopefully obvious. The rover is the black-and-white flea-like critter (it was a rover when the image was larger).



After items are collected to the inventory.



Rover is in a different room. The user has selected to see the portal which leads to home screen



Acknowledgements

This project is based on the work of Lea Wittie, Karl Cronburg, and Shane Markstrum.