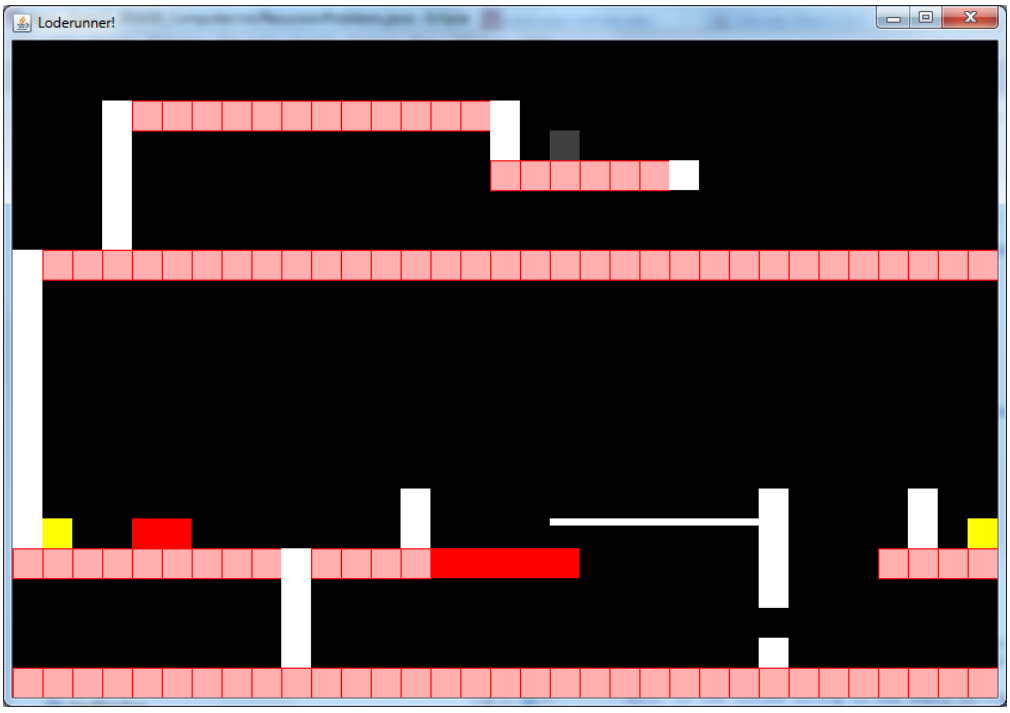
CSSE 220 LodeRunner programming assignment – Team Project

Above is a shot from the real game. Below is a shot from the instructor's solution, which does not use the fancy graphics. You are welcome to add fancy graphics, but this should be the LAST thing you do. It's much better for it to be blocky and functional than to be pretty but do nothing.



Game description: You will write a game that is patterned after the 1990's Broderbund LodeRunner action game. You can find descriptions of the game in these places

http://en.wikipedia.org/wiki/Lode_Runner

A place where you can download and install a trial version (from which the above screen shot is taken):

http://www.zxgames.com/en/loderunner.shtml

Essential features of your program: Your graphics do not have to be fancy (such as figures that look human or that move like they are running, climbing, digging, or falling; actually the hero and guards could be represented by different-colored rectangles, and you could save realistic figures for a "do if we have time" feature). A complete game should have at least the following features:

- A hero who collects gold and is chased by guards (bad guys). He can be controlled by the arrow keys, and can use the Z and X keys to dig (you can also add control by the mouse or by other keys if you wish). Contact with a guard results in death and a restart of the same level, except as described below.
- There are guards who try to prevent the hero from getting the gold and escaping. They move and fall slightly slower than the hero. Each guard moves independently of the others. Your game is *not* required to exactly copy what the guards in the "real" LodeRunner game do.
- The movement of the guards can depend on the positions of the guards, the hero, and the other guards, but it should not be random. On a given level, if in two separate attempts the hero does the same actions with the same timing, the guards should do the same things as well. This predictability will allow the player to use lessons learned in one attempt in order to be more successful in the next attempt.
- Different levels should have different numbers of guards.
- Most contact with the guards causes the death of the hero, but see the sections of the Wikipedia article called "permitted contact" and "trapping and using guards". Your game should implement as much of this functionality as you can. Note that both the hero and other guards can walk on the head of a guard.
- Contact with gold causes it to be removed from the board and added to the score; removing all of the gold on a level causes the appearance of a ladder that allows the hero to escape to the next level. Guards "steal" gold when they encounter it. They only give it up when they are trapped in a hole.
- The board features include bricks, boards, ladders, and ropes. The hero can walk on bricks and boards, and dig through bricks (but not boards). He can climb up or down ladders and move while hanging from ropes. He can fall any distance without getting hurt. After digging, the bricks reconstruct themselves after a minute or so. Anyone who is in that space when this happens dies. If a guard dies, a replacement falls out of the sky.
- You should have at least two levels; each should be "winnable." It is okay to emulate the levels in the original game, or to make up your own.
- A level should be representable by a text file. Such a file can be passed to a Level constructor method to create that level. A level file should include the locations of the hero, guards, gold, and all areas of bricks, boards, ladders, ropes, including the escape ladder that appears when the hero has liberated all of the gold. It should also include the initial locations of the hero and the guards. When the user selects "Play Game", the program should open the Level 1 file, and build the board layout based on what is in that file.
- In the early stages of development, you may want to "hard code" a level, while you are working on the file format and the code for building a level based on a file's contents.
- Pressing the U key should cause the game to go up to the next level; the D key takes you down to the previous level. These features are not in the sample game from zxgames, but they will be very helpful for your (and your instructor's) testing of your game.
- Pause or restart the action by pressing the P key.

Nice features to add. Additional features that you might add include:

- Save the game that is in progress, and load previously saved games.
- High score feature.
- Help screen.
- Animation of sprites that represent the characters.
- Something creative that you want to add.

A major goal of this project: Your team should explore the various classes associated with Java Swing in order to find ways to do various things that you need. We hope this project will help you make the transition from just getting info about classes from the textbook and your instructor to also digging a lot of it out on your own. You may also want to research some general topics, for example animation using Threads (we'll do some of it in the Day 24 class), displaying images, double buffering, and the glassPane (the glass pane may make it easier to draw moving objects without redrawing the fixed background each time something moves). Reading and research may occupy a very significant portion of the time your team spends on this project.

Repository name: `http://svn.cs.rose-hulman.edu/repos/csse220-201320-LodeXY`, where XY is your team number (See this file for list of repository names). Repositories are already created and populated. Each team member should check out the project from this repository, and all subsequent work should be placed in your project folder and committed back to your repository. Don't forget to minimize conflicts by always updating before editing and before committing.

Parallel work: Between now and the end of the term, this project will occupy a lot of your programming time. But there will still be a few daily programming assignments along the way.

User stories: This is a simple way of capturing the program's desired functionality. As a Google search will quickly attest, there is much variation in what various people mean by a user story. Basically it should describe what the user (or in this case possibly one of the characters on the screen) might do and what happens as a result. Each "story" begins with something the user (or an actor in the program) does, and ends with a description of what the program does (or computes) as a result. Here are some examples related to LodeRunner:

1. User presses right arrow key. The hero begins to move right, unless an obstacle prevents this movement.
2. Moving actor encounters a rope. Keeps moving, hanging from rope.
3. User presses X key. If there are bricks to the right of where the hero is standing, some bricks are dug out.
4. User presses down arrow key when hero is on a rope, or hero moves into empty space. Hero falls until an obstacle (e.g. board, bricks, rope, guard) is encountered.
5. When hero encounters a rope while falling, user presses left or right arrow key. Hero grabs the rope.
6. Hero encounters gold. Gold disappears and score increases. If it is the last gold on the level, escape ladder appears.

Writing user stories is part of the analysis phase of software development; it should be one of the first things you do. It is a simple way of discovering the things that the program should do. The words in the user stories should help you to discover classes and responsibilities.

When it is time to enumerate the user stories you plan to implement for the next development cycle (see the next section), you can mostly pull them from your "master list," although things that you learned while doing the last implementation cycle may cause you to add or modify some stories.

Development cycles: You will do this program in several short development cycles, most lasting three or four days; the last one only one day. Before the beginning of each cycle, you will list some user stories that describe what functionality should be present at the end of that cycle.

Milestones (all due at the beginning of class, except as noted):

Key points:

1. To get credit for the milestones, every student should have submitted code (I estimate at least 50 lines per person)
2. The code checked into your source control must work (i.e. should compile and run directly from source control with no special tricks)
3. The code should implement significant new functionality (as specified by your stories)

1. Wednesday, May 7th (Day 23)	UML class diagram, CRC cards, user stories, select stories for Cycle 1, begin coding Cycle 1
2. Friday, May 9th (Day 24)	Cycle 1 code and progress report, user stories for Cycle 2
3. Friday, May 16th (Day 27)	Cycle 2 code and progress report, user stories for Cycle 3 (adjusted because it was wrong on the website)
4. Monday May 19th (Day 28)	Cycle 3 code and progress report, user stories for Cycle 4
5. Friday, May 23rd (Day 30)	Final Code and documentation (see grade components below), Project Demo in class
6. Tuesday, May 27th 1pm (Finals	Team member evaluation survey (required if you want a grade for this project).

Week)	

Milestone 1: UML Class Diagram and **CRC cards**

You should go through the same kind of process that we used in the in-class Email exercise:

1. Write your user stories.
2. List the responsibilities of the LodeRunner program (We would guess that you will find between 20 and 40). One way of finding responsibilities is to develop some user stories (see below).
3. Brainstorm possible classes. (We would guess that you will come up with about 8-12 classes)
4. Assign responsibilities to classes; determine how classes need to collaborate in order to carry out those responsibilities, and what responsibilities those collaborating classes need to have. Will inheritance or interfaces help you to organize the responsibilities? Keep iterating this until all of the program's responsibilities have been assigned to classes.
5. Collect the information into a UML class diagram.
6. Turn in in your CRC cards at the beginning of Day 23 class. (You should also get some code written before that class)

Save your diagram as a PDF or JPG file, so it can be viewed without UMLet. Put both the (PDF or JPG) and UMLet files in your repository, in the `DesignAndProgressReports` folder.

Once you have documented your classes on the UML diagram, select user stories that you plan to implement in the first cycle (before Day 24). Complete the `Cycle_1_Plans.txt` document and commit it.

Begin implementing, commenting, and testing your code, cycle by cycle.

Document your code as you go along.

Subsequent Milestones:

Before class on the day milestone K begins, complete the Cycle_K-1_report_and_Cycle_K_Plans.txt document, where you will discuss what was accomplished in the previous cycle and what you plan to accomplish in the coming cycle. In particular, which user stories will you implement? Who has done the primary work on different things that the team has accomplished, and what are everyone's assignments for the next cycle?

Create JUnit tests for any parts of the program that can be tested without the GUI.

Commit your project often.

Teamwork and grading:

This assignment will be done by three-person teams. If the number of students in your section is not a multiple of three, there may be one or two teams of two students. Our intention is not that you "divide and conquer" so much as that you have someone to talk with as you write and test this program. If you have not already done so, read this short article on Pair Programming and discuss it with your partners: http://en.wikipedia.org/wiki/Pair_programming. In particular, note what it says about who should be the driver if you are a "mismatched pair".

All code that you submit for this project should be understood by all team members. It is your responsibility to (a) Not submit anything without first discussing it with your partners, and (b) not let something your partners write go "over your head" without making a strong effort to understand it, including having your partners explain it to you of course.

This project should give you practice with the "short cycles and user stories" approach to software design and implementation.

It is possible that different team members will receive different scores for the project, if there is ample evidence that one person did not fully participate in the learning and the doing (or that one person "hijacked" the project by insisting on doing most of it without much help or understanding from the rest of the team), We reserve the right to give different grades. A peer evaluation survey at the end of the project will help us to determine this. If the survey or our observations indicate that you do not understand, we may ask you to explain parts of your project code to us.

We will expect your evaluation of your team members at the end of the project to be detailed and specific. You should be writing it as you go through the project. Make notes of both positive things and suggestions for improvement. Then when it is time to submit your evaluation, you can mostly just paste what you have written into the ANGEL survey.

Grade components:

15 points	CRC cards and initial UML diagram
10 each	Milestone reports for Cycles 1, 2, and 3
25 each	Code fuctionality for Cycles 1, 2, and 3
125	Final program functionality and correctness
50	Style and efficiency
25	In-class presentation
25	Thoughtful team evaluation and reflection on the project (individual)
??	Additional features (extra credit)

Disclaimer: This document may be revised in response to student questions/corrections. The latest version will be considered the authoritative one. If any changes significantly modify or clarify the project requirements, we will notify all students by email, to make sure that you read the new version of this document.