Eric Olson
Assignment Report – Final Project
CS162 – Fall 2014 – Rooker

Understanding

The final project combines themes of a number of prior assignments and labs. The overall design is intended to be a text based game in which the player will navigate a set of rooms, collecting items to accomplish some goal. More specifically the game needs to implement a layout of linked room objects (at least 10) that can be navigated by following pointers. The game must also contain a 'bag' of some size in which objects can be collected necessary for solving the overall goal. The items must 'fit' in the bag and if desired items would overfill the bag items should be able to be dropped or left behind. There should also be some elements that help the player (and grader) solve the puzzle along the way.

Specifically the game elements should include:

- **Linked Rooms Abstract Class** – An abstract room class should be developed with the majority of the data elements, and virtual abstract functions for what actions can be performed in the child room classes.
- **Child Room Classes–** At least 3 child room classes need to be developed that add some unique action or features to the parent room class.
- **Puzzle –** There should be some stated goal that players are working to solve while traversing the rooms.
- **Bag –** There should be a bag of a specified size. Objects collected in the various rooms can be stored in this bag but take up a defined amount of space. When the bag is filled objects may need to be left behind to make room for other items.
- **Hints –** There should be helpful options along the way that allow the player/grader hints to solve the puzzle.

Program Design

This design is being drafted prior to any coding. Once programming has begun this design may need to be altered to allow for some unforeseen issue. Any key changes will not be reflected in this design but in the reflections section at the end of this report.

I've chosen to develop a riddle game on a train. I will come up with some back story in which the player is a passenger on a train that has been hijacked in some way. The player will be responsible for solving a riddle in order to save the train. In order to solve the riddle the player will need to move through the train looking for pieces of the riddle, as well as letters that can be used to solve the riddle. Once the riddle is solved and the player has collected the required letters they can solve the riddle at the front of the train.

As I draft this program's design I'm considering a few key items, that include: Room Design, Train Layout, Helpful Rooms, Bag & Letter Storage and Riddle.

Room Design

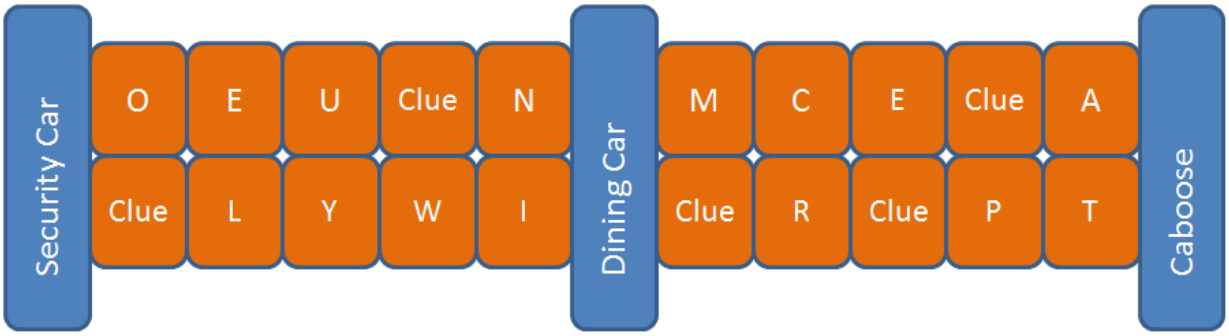
There should be a parent abstract class with a few key items. The data elements should include room contents, and pointers in all 4 cardinal directions. Member functions should include a virtual action() function that can define the particular action of the child rooms, a move() function and a setup() function.

The move() and setup() function can be defined at the parent class level as all activity would be the same for all classes. The setup() will layout the train design and add contents to each room. The move() function will have the user select a direction to move, and move the player in the chosen direction.

I plan to use 4 child room classes, one for a Caboose, a Dining Car, a Passenger Car and one for a Security Car. The Caboose will be where the user begins their quest and can explain the game. The Dining Car will be a room in which the player can find out how many of their collected letters are correct, meaning used in the final solution to the riddle. The security car will be where the player finishes the puzzle, and may contain a hint of its own, perhaps confirmation of a specific correct letter if the user is not ready to solve the riddle. The rest of the cars will be passenger cars where letters or riddle components can be found or left.

Train Layout

I plan to use 10 passenger cars, and one each of the other types of cars. The train front to back will be a Security Car, 5 Passenger cars (left and right side accessible separately), the dining car, 5 more passenger cars and finally the caboose. I've also laid out the letters and clue pieces in each section as follows:



When leaving a specialty car the path will default to the left, you can enter a specialty car from either the right or left side of a passenger car.

Helpful Rooms

The Dining car can be used to help players by giving them a count of the number of letters in their bag that are also in the final riddle answer. The actual letters won't be identified, but the # of letters will be.

The Security Car in addition to being the place that allows you to solve the riddle will also confirm one of the correct letters you have in your bag by name. Or conversely will let you know if you have no correct letters.

Bag and Letter Storage

I will define a bag class that will hold letters. Each letter will have a value or size associated with it. I can use a pair container out of the STL to house these two values (char/int). The bag will have a size defined by the value of all the correct letters in the final answer. There will be additional letters and some with large values so situations will arise where the player will not be able to pick up a given letter due to lack of room in their bag. In those situations the player can attempt to swap a letter out of their bag, or go to an empty room and drop off a letter. No more than 1 letter can be left in a room at a time.

Riddle

I have selected the following riddle:

There was a green house.
Inside the green house was a white house.
Inside the white house was a red house.
Inside the red house were lots of babies.
What is it?

Watermelon

I chose this riddle for a few reasons, I don't believe it's well known, and it has a single world answer with many letters, but almost all unique letters. Additionally the riddle itself is easily broken into 5 sections that can be hidden throughout the train and uncovered line by line to further the mystery. As a clue piece is found it will simply be added to the known riddle and leave the car empty.

Program Testing

While some of this functionality was tested with previous labs/projects it was implemented here in new ways (struct vs. class, STL container vs. array etc). Therefore I've pulled in many of the tests performed on the original designs, as well as a large number of new things to check unique to this program. Items in this list were collected both prior to coding, and during the development phase.

Test	Expected Result	Actual Result	Correction
Room Design & Navigation			
Able to move 4 Directions	When appropriate able to move	As Expected	N/A
Restricted when NULL	Unable to move when direction = NULL	As Expected	N/A
Rooms type in correct layout	Each room correctly laid out and connected	As Expected	N/A
Rooms containing objects	Rooms contain correct letter/clue	As Expected	N/A
Special Cars In/Out	Able to enter from Left or Right, exit to left.	As Expected	N/A
Caboose			
Game Intro	Provides Game Intro	Found to be annoying since room could be re/entered.	Moved Game intro outside of Caboose.
Passenger Car			
Contains a letter	Prompted to pick up letter	As Expected	N/A
Contains a riddle	Riddle added to overall riddle	As Expected	N/A
Empty	Offer to drop a letter from bag	Also offered when bag was empty.	Added check to ensure at least 1 letter present.
Dining Car			
Counts Correct Letters	Counts Correct Letters	As Expected	N/A
No letters present	Correctly says 0 letters correct	As Expected	N/A
Security Car			
Incorrect Letter	Correctly identifies incorrect letter	As Expected	N/A
<10 correct letters	Correctly identifies correct, but not complete.	As Expected	N/A
10 correct letters	Enters solving loop	As Expected	N/A
No letters present	Treats like <10 correct.	As Expected	N/A
Solving Loop - Correct	Accepts correct answer	As Expected	N/A
Solving Loop - Incorrect	Let's user try again	As Expected	N/A
Not Case Sensitive	Accepts lower case, or mix of cases	Originally only accepted upper case	added toupper() logic.
General Turn Mechanics			
Location Display	Should show current location	As Expected	N/A
Riddle Status	Should show unlocked riddle	As Expected	N/A
New Riddle Piece	New Riddle piece should show appended to riddle	As Expected	N/A
Bag Contents	Should show letters / size correctly	As Expected	N/A
Action Taken	Should call car specific action()	As Expected	N/A
Move when finished	Should call move() when action() is complete.	As Expected	N/A

Program Testing (cont)

Test	Expected Result	Actual Result	Correction
Bag Mechanics			
Accept letter w/ space	Letter should be added to bag, space updated correctly.	As Expected	N/A
Accept letter w/o space	Swap functionality should be called	As Expected	N/A
Swap with room	New letter accepted and old letter returned to car	Issues with return passing.	Changed function to always return what should remain in car.
Swap with no room	No change to car or bag.	As Expected	N/A
Drop Letter (empty room)	Function should show size next to letter, and selected letter dropped.	As Expected	N/A
Bag order	Letters should be in order picked up.	As Expected	N/A
User Input			
Move()	Should only accept 1-5, and no letters	As Expected	N/A
Letter action (y/n)	Should consider first letter entered, and repeat if not Y/y or N/n	As Expected	N/A
Drop/Swap letter	Should only accept held letters, not case sensitive	issues with case	added toupper() logic.
Final answer	Should only accept in correct order, any case.	issues with case	added toupper() logic.

Reflection

- Overall this program went together well, and benefited from the previous labs/assignment work that shared similar components, functionality.
- I did have to make a number of changes to the original design in order to solve for issues usually playability related, or specific to passing variables to functions, as outlined below.
- I didn't like having the Caboose contain the intro text, as a user could re-enter the caboose and would again be presented with the intro scenario. I moved the contents outside the caboose to solve for this.
- I had considered having the Security Car provide a hint of 1 correct letter in your bag, but this wasn't a repeatable helpful tool as it would always provide you the same letter. I therefore changed the hint to be one incorrect letter, or confirmation that all letters were correct. This way the player could go drop the identified incorrect letter and return expecting a new hint.
- Originally I had the setup() function as a member function of car. In theory this made sense, but I needed to be able to call the function from a pointer as in car->setup(), and change the pointed to address of the calling pointer which I was unable to do. I moved setup() into a friend role, and passed it the pointer and this solved my issue.

- Per the same issue above, I had to move the move() function in order to change the location to which the player pointer was pointed. I passed move() the pointer variable and this quickly solved the issue.
- I ran into some inefficiencies with the action() abstract function. In some of the cars I needed the bag contents and clue details and in others I did not. In order to have the functions redefined in the children classes they all needed the same function signature, which required that I add the passed variables to the parent class action() function and then to all of the children classes action() function as well. That was a lot of copy and paste amongst all the headers/trailers, and would have benefited from some more upfront design foresight.
- I struggled for a while as to how to indicate that my game was complete and exit the main() loop. I ended up changing all the action() functions to return a bool value so I could easily check to see if the game was complete. Per above again this required a change to every single action() function in every header/implementation file.
- In the end I was pleased with how the program turned out. It won't win any gaming awards at next year's conventions, but it met the requirements criteria and worked as well as I had hoped. I look forward to leaving text based programming and working with GUI's and graphics!