Peter Moldenhauer
moldenhp@oregonstate.edu
11/26/16

CS 162 Final Assignment Reflection

Project Requirements:

- Design and implement a text-based game where the player moves through a series of rooms or spaces.
    - Each space will be a class with at least four pointer variables that link to other spaces
    - Any unused pointers will point to NULL
    - There must be at least six spaces of at least three different types.
    - There will be a space abstract class that will have pure virtual functions.
    - There must be a way to keep track of which space the player is in.
- There must be a goal for the player in the game.
- The player will have a container to carry "items".
    - One or more of the items will be required as part of the solution
- The player must interact with parts of the space structure
    - When the player is moving around, he will not only get information from the spaces but also change the status of the spaces.
- Provide a menu option for the game
- Before the beginning of the game, reveal the goal for the player
- Develop a theme for the game

Design Plan:

The overall goals of this assignment are to design a program that satisfies the above requirements in addition to using good OOP style, using inheritance, using polymorphism and using pointers. Other than the requirements listed above, all of the other implementation details will be left up for me to decide. In order to complete all of the goals and requirements, I will create a text-based adventure game which has a train theme. In this adventure game, the character will move throughout a train to look for their lost phone. Along the way, the character will talk with other passengers, interact with the environment and gather objects that allows for progression through the train. The game will take the form of a puzzle, with clues providing evidence for how to proceed. When the character finally finds their cell phone in the train, the game is over and the user won the game.

The train environment (i.e. spaces) will be made up of various train cars, people on the train and rooms in the specific train cars. The structure of the train will be linear so the front of the environment will be the front of the train (locomotive) and the back of the environment will be the back end of the train (baggage car). When the character is in a

specific train car, the character will also be able to move left and right to move to people in the train car or different rooms in the train car. There are a few different visual representations of the train environment included in this document to give a better understanding of the layout of the train.

Description of Classes:

This program will be made up of three main classes: Space, Character and Train. In addition to these classes, the main.cpp file containing the main function will run the game. To create the game environment, all Spaces will be instantiated in Train and pointers set to adjacent spaces. Each Space will contain a pointer to the Train's main Character object. In main.cpp, Train will first be instantiated, then its character object will be stored in a pointer in main.cpp to use in the game. A function *gameLoop()* will be used in main to run the game. Exiting the game will exit the loop. Below is an overview of these various classes that will make up the game:

Space:

The assignment requirements call for an abstract class from which all of the locations in the game will inherit from. The Space class will be the abstract class in my train game. Pointers will then be used to connect different Space objects together. In this game, each Space will have fwd, back, left and right pointers. The Space objects will allow for movement in four directions while edges will point to NULL. To interact with the environment, the abstract class Space will contain three virtual functions (two of which are pure virtual): talk(), lookAround() and manipulate(). Each of the derived classes from the Space class will then implement their own versions of the talk(), lookAround() and manipulate() functions. The abstract base class Space can be seen here:

Space
protected member variables: Space *fwd Space *back Space *left Space *right Character *character std::string spaceName

public member functions:
Space() //constructor
void setLocation(Space *f, Space *b, Space *l, Space *r, Character *c, std::string n)
Space *getFwd()
Space *getBack()
Space *getLeft()
Space *getRight()
Character *getCharacter()
std::string getSpaceName()

void setSpaceName(std::string) virtual void talk() virtual void lookAround() virtual void manipulate()

There will be numerous derived classes that inherit from the Space base class. Each of these derived classes will implement their own versions of the three virtual functions talk(), lookAround() and manipulate(). Most of the dialogue and interaction will be contained in these different Space derived classes. All of these derived classes linked together is what is used to make up the Train environment for the game. These derived classes are linked together with pointers in a specific manner to each other to provide a structured layout of the train environment. The structure and linking of these derived classes can be seen here:

Class Name	Description
BackRestroom	Rear starting point for the character at the back of the train
BackLoungeCar	Rear train car space with forward, back, left and right pointers
SnackStand	Space pointed to by BackLoungeCar where the first interaction occurs.
BaggageCar	Last train car space that is initially locked. This space is pointed to by BackLoungeCar where game ends.
CoachClassCar	Train car space pointed to by BackLoungeCar with forward, back, left and right pointers
CryingBaby	Space pointed to by CoachClassCar where interaction occurs
SnoringSleeper	Space pointed to by CoachClassCar where interaction occurs
FirstClassCar	Train car space pointed to by CoachClassCar with forward, back, left and right pointers
BradPitt	Space pointed to by FirstClassCar where interaction occurs
DonaldTrump	Space pointed to by FirstClassCar where interaction occurs
ForwardAttendantCar	Train car space pointed to by FirstClassCar with forward, back, left and right pointers
FrontGalley	Space pointed to by ForwardAttendantCar where interaction occurs
FrontRestroom	Space pointed to by ForwardAttendantCar where interaction occurs
LocomotiveCar	Train car space pointed to by ForwardAttendantCar with forward, back, left and right pointers. This space is locked until the character gains access
ConductorSeat	Space pointed to by LocomotiveCar where interaction occurs
EngineerSeat	Space pointed to by LocomotiveCar where interaction occurs

LookingOutside	Space pointed to by LocomotiveCar where interaction occurs

Character:

The character object will represent the game's main character. The character object will have a *name* string member variable, and a vector of strings to hold objects acquired during gameplay. Member functions will include moveFwd(), moveBack(), moveLeft(), and moveRight(). A *currentSpace* member variable will also be included to represent the characters current space. The character class can be seen here:

Character
protected member variables: std::string name Space *currentSpace std::vector<std::string> objects

public member functions:
Character() // constructor
void setName(std::string)
void setCurrentSpace(Space *s)
Space *getCurrentSpace()
std::string getName()
bool objectSearch(std::string o)
void moveFwd()
void moveBack()
void moveLeft()
void moveRight()
void addObject(std::string o)

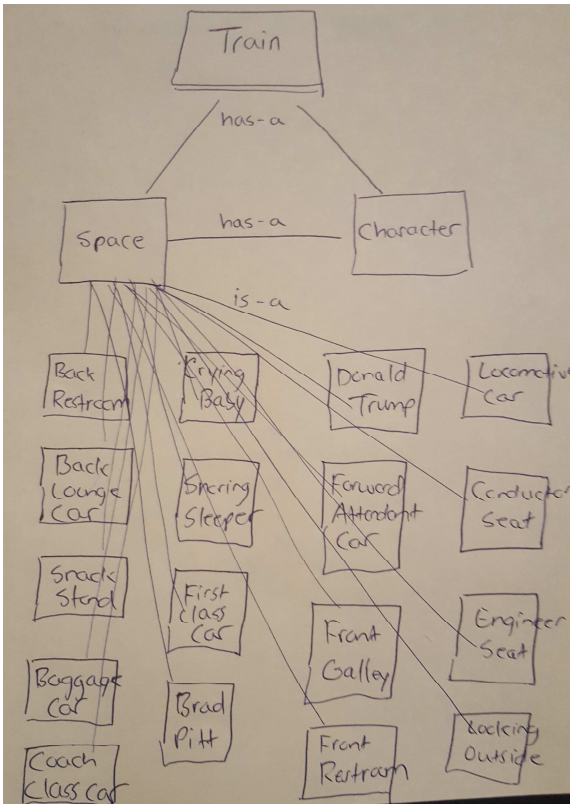
Train:

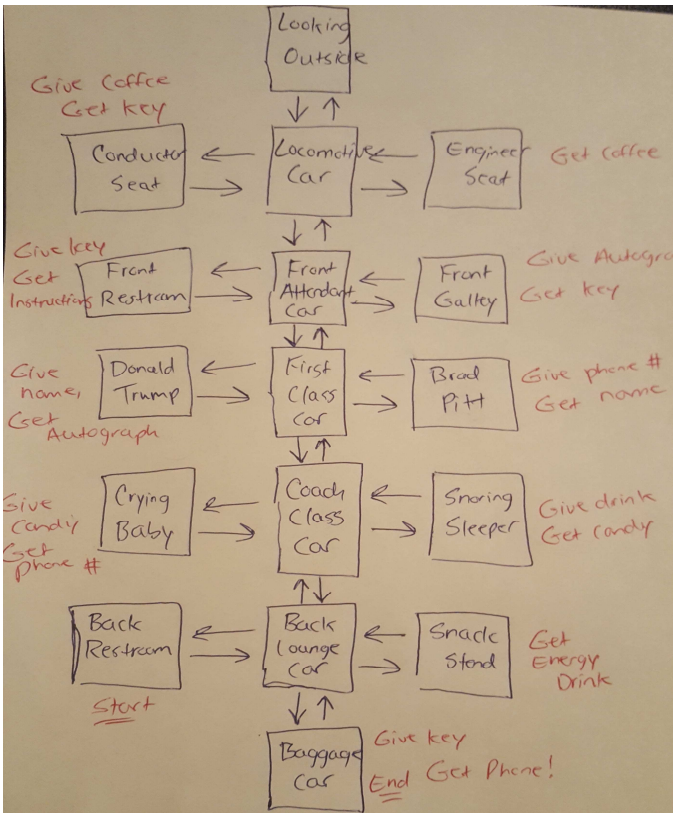
To limit the size of main.cpp, I will use the Train class to manage all of the Space and Character class objects in the game. A Train object will be created to instantiate all of the Spaces along with Character. Also, inside of the Train class, the pointers between Space objects will be set up to create the environment or "layout" of the train. The Train class will also hold the primary pointer to the Character object, which will then be used in each Space. The Train class can be seen here:

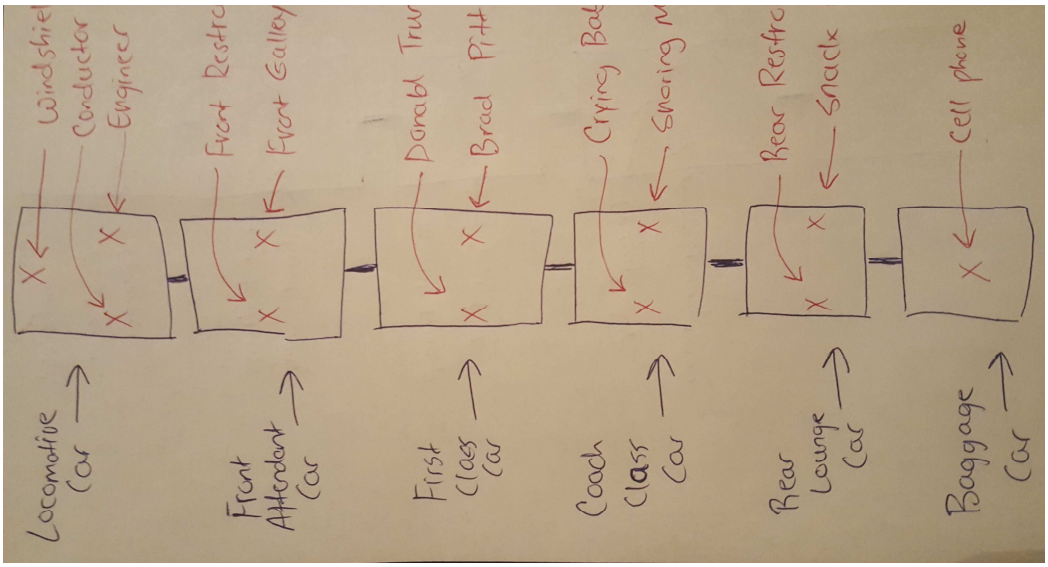
Train
private member variables: Character *character

public member functions:
Train() // constructor
Character *getCharacter()

Visual Representations of Train environment:







Game Play:

The following steps explain the solution to winning the game. The talk() and lookAround() functions enhance game play for the user by providing clues, but the manipulate() function is the only function necessary to move through and win the game.

1. Character starts in Back Restroom. Baggage Car and Locomotive Car are locked.
2. Move two spaces right to the Snack Stand and get the energy drink from the train attendant to wake the snoring man.
3. Move left, then one space forward to the Coach Class Car
4. Move right to the Snoring Sleeper and give him the energy drink. Gain candy from the Snoring Sleeper to stop Crying Baby
5. Move two spaces left to Crying Baby and use the candy. Gain Angelina Jolie's phone number.
6. Move one space right and one space forward to the First Class Car.
7. Move one space right to Brad Pitt. Give him Angelina Jolie's phone number. Get the front train attendant's name.
8. Move two spaces left to Donald Trump. Give him the front train attendant's name and gain an autograph.
9. Move one space left and one space forward to the Front Attentants Car.
10. Move one space right to the Front Galley. Give the train attendant the autograph to gain a key to use in the Front Restroom.
11. Move two spaces left to Front Restroom. Use key to gain instructions to access the Locomotive Car
12. Move one space right and one space forward to the Locomotive Car
13. Move one space right to the Engineer Seat. Gain coffee to give to the Conductor
14. Move two spaces left to the Conductor Seat. Give conductor coffee and gain a key for the Baggage Car.
15. Move one space right and all the way back to the Baggage Car.
16. Talk with person in the Baggage Car. Answer to the riddle is "nothing".
17. Game ends

Testing Plan:

My overall testing strategy for this assignment was to use "incremental development". More simply, I proceeded to test my program continually as I wrote the code for it. I got one part of my program written out and then I tested it right away. If the program compiled and ran properly, I would then proceed to write another part of the program. If this part of the program did not compile and/or run properly, I would work out the bugs in this one small segment of code before even starting on writing other parts of the program.

My plan was that by using this incremental development strategy, a few things would work in my favor. First, by writing only small portions of code at a time and testing right away, if an error did exist then I would only have to look at small blocks of code at a time

to fix the errors. If I wrote out the entire program before testing, I am sure it would be very time consuming and frustrating going through the entire program top to bottom over and over again until I found all of the errors. Second, by continually testing my program over and over again, I not only am able to make sure everything runs properly but also that the output to the user displays as intended. I feel that adding spaces and new lines throughout this testing phase to make the output look presentable was less time consuming versus waiting until the very end to go through the program all over again to format the output properly. Lastly, using incremental development ensured that I had at least some sort of a working program to submit at all times. If something were to happen in which I did not have enough time to finish the program before the due date, at the very least I could submit what I have already completed. In this situation, my program may not be complete but I know that it runs properly so I would most likely be able to receive at least partial credit for the assignment.

For the specific tests for this program, I basically have to test every possible movement of the character in the train environment along with testing every possible interaction the character can have in each environment space. In other words, I have to test that the character is allowed to move to each space and not allowed to move off the boarder of the train environment. For example, with the starting location (Rear Restroom) I have to make sure that the character is only able to move right (to move outside of the restroom) and not able to move forward, back or left while still inside the restroom.

I also need to be sure that the character is not able to completely advance in the game without interacting with various parts of the environment first so that other areas of the environment will be unlocked. For example, I need to make sure that the character needs to get the key to the baggage car first before entering the baggage car to find the phone. The character should not be allowed to simply start the game and enter the baggage car.

I also have to test to make sure the character in the game is facing the correct direction each time. I plan to have the character facing forward (towards the front of the train) in every single Space which should not change. For example, if the character starts in the rear restroom and moves *right* to get to the rear lounge car, the character should be facing forward in the rear lounge car. So then to get back into the rear restroom the character would then have to move *left* (not *back* which might make more sense but this would complicate the program more than it needs to be).

Lastly I need to test to make sure that the user does indeed win the game when the character enters the baggage car and finds the phone. This is important because if the game is not won when the character finds the phone in the baggage car, this program might turn into an infinite look of moving around in the train. Similarly, I need to test to make sure that each one of the menu functions work as intended, specifically to exit the program. Since this game is fairly long, the user should be able to exit the program at any time. To account for this, the user is able to push ‘5’ at the menu at any time to exit the game.

Test results and how problems were resolved:

Overall the testing process of the program went fairly smoothly. The most difficult problem I ran into was figuring out how to properly move the character between spaces. My original plan was to have a charPresent value that would set to true as the character moved to each Space. However, I found that it was much easier to have every space point to the same Character object, and change the currentSpace value in the Character object accordingly.

My initial plan was that Space contains a Character object member variable, and Character a Space member variable. However, this would not compile at first. After some research online, I learned that the header files had to imported into each class's .cpp files with a "forward declaration" in the .hpp files to achieve this. This was my first time hearing and learning about forward declaration.

In my final copy of the program, I added Boolean variables to the Character object to represent the keys for locked doors. I also added additional logic to moveFwd() and moveBack() to prevent movement into these areas without hasKey and hasCKey being set to true. Originally, I had added these Booleans as part of the Spaces for which access was prevented.

Below is a chart of the final testing results for the allowable movement and interaction in each Space of the train environment:

Space	**Allowed Movement**	**Allowed Interaction**	**Result**
BackRestroom	Move right only	No interaction	As expected
BackLoungeCar	Move left, right, forward at start of game, able to move back when BaggageCar key is obtained	No interaction	As expected
SnackStand	Move left only	Talk to train attendant, obtain energy drink	As expected
CoachClassCar	Move left, right, forward and back	No interaction	As expected
SnoringSleeper	Move left only	Talk to sleeping man, give energy drink, obtain candy	As expected
CryingBaby	Move right only	Talk to sister, give candy, obtain phone number	As expected
FirstClassCar	Move left, right, forward and back	No interaction	As expected

BradPitt	Move left only	Talk to Brad Pitt, give phone number, obtain name	As expected
Donald Trump	Move right only	Talk to Donald Trump, give name and obtain autograph	As expected
ForwardAttendantCar	Move left, right and back, able to move forward when the LocomotiveCar instructions are obtained	No interaction	As expected
FrontGalley	Move left only	Talk to train attendant, give autograph, obtain key to FrontRestroom	As expected
FrontRestroom	Move right only	Unlock panel in restroom, obtain LocomotiveCar instructions	As expected
LocomotiveCar	Move left, right, forward and back	No interaction	As expected
EngineerSeat	Move left only	Talk to engineer, obtain coffee	As expected
ConductorSeat	Move right only	Talk to conductor, give coffee, obtain BaggageCar key	As expected
LookingOutside	Move back only	No interaction	As expected
BaggageCar	Move forward only	Talk to person, solve riddle, obtain cell phone	As expected

Final Thoughts:

I enjoyed the flexibility that I was allowed to have for this final assignment. Even though there were a few basic requirements, the majority of this assignment was left up to my own creativity and implementation to solve the problem. It helped tremendously by writing out a design plan and having a rough design to work off of when I started to code. Having an idea of how all the classes fit together was also very helpful when I began to code the program. This program focused heavily on the use of polymorphism and virtual functions so having a rough plan of how all of the Space classes were connected was a

big help. Overall, my design matched my final program fairly well. There were only a few relatively small changes and errors that I had to spend some extra time on.

I thought this was a fun project and was a great way to build my knowledge and experience of polymorphism, pure virtual functions and pointers. I am excited to start building my own games similar to this one in my free time.