Jeffrey Schachtsick

June 1, 2015

CS162: Final Project

**Understanding**

This is the Final project for the Introduction to Computer Science II class. In this project, I will be developing a text-based game where the player moves through different locations toward achieving some goal. The purpose of this project is to satisfy all the requirements and to use good object-oriented programming style, which include inheritance.

As part of the requirements, I will need to have create a series of locations for the player to move through. There should be more than 10 of these locations in a linked list of sorts, and the player should be able to traverse between these locations to achieve the objective. I've decided in my program to create a race game, in which the player will be presented with a random set of locations, take their car, traversing the course to get to the finish line. However, just like in the fictional story of *Speed Racer* there were will be obstacles and quests. The obstacles will either force the player to go backwards or stall their ability to move forward. The quests are little more than side quests, where the player can move forward or backward to getting closer to the end-goal or getting a small reward.

The second requirement, is to have the player complete some end-goal. I sort of mentioned the end-goal above, but as an added couple goals I will add that the player does not race by him/herself as there is an adversary who also races. The player needs to beat the time of the adversary and as an added bonus, beat the top ten recorded times.

Another requirement is to keep track of which location the player is in. While another requirement is to have the player be able to carry items. Each car should have some storage, where if the player gets into trouble, can use an item to bypass the danger. Should the player not have the item, then there should be a penalty involved.

**Testing**

| **Test** | **Description** | **Input** | **Output** |
|---|---|---|---|
| Start Game | At game start, Race Track is created. | User is given a Main menu. User agrees to start a new race | Track is created with random locations and given a race name. |
| Continue Game | When a game has come to an end, user can select to race again | User is given a Main menu. User agrees to replay last race. | Previous track is used again with same locations. |
| Select a car | User selects a car. Should show as the object throughout game and comes with attributes for storage and passengers. | User selects RallyCar | RallyCar should show attributes with number passengers and storage available. Turns per location and gas |

| | | | |
|---|---|---|---|
| | | | consumption ratio should also be factored as the game goes. |
| Contract Ninjas | User is at the shop for upgrades and selects to contract ninjas | User selects to contract ninja for RallyCar | RallyCar had an empty seat prior to contract, but now the seat is occupied with a ninja. Credits for contract is subtracted from player's total |
| Too many passengers | User selects to add another passenger, but vehicle is already full. | User selects to contract ninja for RallyCar | Error displays as there is already max capacity in vehicle. Credits does not decrease. |
| Purchase supply | User is at the shop for upgrades and selects a supply. Vehicle has enough space | User selects a gas can for vehicle | Gas can is in inventory. Credits for contract is subtracted from player's total |
| Too many supplies | User is at the shop for upgrades and selects a supply. Supply would take over the amount of available inventory. | User selects a gas can for vehicle | Error displays as there is not enough. Credits does not decrease. |
| Player gets a danger card. Evades. | During race, player gets a danger card. Player rolls to evade danger. Evades danger | Danger card given, player must roll between 3 and 5 to evade penalty. Rolls 4. | Danger evaded. moves to next turn. |
| Player gets a danger card. Pays penalty | During race, player gets a danger card. Player rolls to evade danger. Pays the penalty of loss of turn. | Danger card given, player must roll between 3 and 5 to evade penalty. Rolls 7. | Penalty given to player. Must validate player receives punishment (i.e. loss of funds, turns, etc.) |
| Fill gas tank | After danger card, Player elects to fill the gas tank before ending turn. | Player selects to fill the gas tank with gas can from supply. | Vehicle now shows it has a full tank. |
| No gas left. | Before danger card, player has no gas left. Must not proceed forward to next location or iteration. Must have penalty for finding gas. | Player ends turn with not enough gas to make it to next location. | Gas will not make it to next location, find gas for the next 3 turns and any additional penalty. |
| Player gets a quest card. | After danger card, player elects to take a quest card. | Player elects to take quest card. | Player has quest card in inventory also must show passenger seats are taken or inventory slots are taken. Displays location of where to meet goal of quest. |

| | | | |
|---|---|---|---|
| Player gets a quest card. No room in inventory. | After danger card, player elects to take a quest card. Has no room to add quest item to inventory. | Player elects to take quest card. | Error: No room in inventory. |
| Player completes quest | After danger card, player arrives at quest destination. Confirmation of achievement and any additional award added along with subtracting quest items. | Player reaches quest destination. | Congrats! Quest Destination made. Receive award with additional items to inventory |
| Player completes quest, no room in inventory | After danger card, player arrives at quest destination. Confirmation of achievement and no additional award added along with subtracting quest items. | Player reaches quest destination. | Congrats! Quest Desitination made. Error, you have no more room in inventory. |
| Player completes quest, receives special item | After danger card, player arrives at quest destination. Confirmation of achievement and special item is added to special attribute to vehicle. Or replaces the current one. | Player reaches quest destination. | Congrats! Quest Destination made. Receive award with additional items to inventory, or special item. |
| Player completes race | Player reaches the finish line. Details total time to complete, time for the adversary to complete, and the top 10 scores. | Player reaches finish | Congrats! You have completed the race. Your score <score>, The adversary's score <score> . Then list 10 ten scores from some input file. |
| Player does not evade danger, plays inventory item. | Player plays danger card. Attempts to evade with roll but fails. Uses inventory or special to evade. | Player gets danger card, cannot evade. Player selects to use inventory. | Show that player moves forward with no penalty from danger, but only subtracting from inventory. |
| Player does not evade danger, plays inventory item that is not there. | Player plays danger card. Attempts to evade with roll but fails. Uses inventory or special to evade that is not there. | Player gets danger card, cannot evade. Player selects to use inventory. | Error: This item is not in inventory. |
| Player does not evade danger, plays special item that is not there. | Player plays danger card. Attempts to evade with roll but fails. Uses special to evade that is not there. | Player gets danger card, cannot evade. Player selects to use special item. | Error: You either have no special item or special item does not evade this danger. |
| Player elects to not take quest card | After danger card, player elects to chance quest card. Chooses not to take it. | Player selects to take a quest card. Elects to not accept this quest | Validate quest is not taken. No quest items are added to inventory |

| | | | |
|---|---|---|---|
| | | | or passenger seats. |
| No money, no gas | Player has no money and no gas. Must work to fill the tank with extra credits | Player ends turn with not enough gas to make it to next location. | No money and no gas! Lose turns to work. Loss of turns is seen, gas tank is full, extra money. |

**Design**

```
Main Implementation file
Welcome the user!
Establish game rules
Loop (Until user wants to exit)
- User can choose:
    - Start race.
        - If greater than first iteration.
            - Ask if to keep current track?
                - No?
                    Create track.
        - Else
            Create track.
        - Loop (Until user agrees to select) Menu of cars.  User selects a car.
            - Are you sure?
        - Loop, menu of shop items ( Until user elects to exit shop items)
            - User selects items to add to inventory.
        - Loop, Until user crosses finish line.
         - Traverse to next location.  Move backwards or forwards.
        - If location is Half-way Station?
            - Menu supply and contract shop
            - increase one turn.
            - continue next iteration of loop.
            - Loop, Until distance = speed.
                - If gas ran out,
                    decrease distance, or if distance is less than 0, go back to last
                    destination with speed – 1 for distance.  Pay the penalty for
                    running out of gas.
                - Play the danger card.
                - Menu to ask player to do one of the following:
                    - Play the special ability. (if applicable)
                    - Chance the quest card.
                    - Fillup gas tank.
                    - Move forward or backward?
    - View top scores
```

```
        - If first iteration:
            - Read input file to array and display to output.
        - Else
            Read array.
    - Exit program
        - Save top score array to output file.
Exit the program.
```

<u>Carr Header and Implementation Files</u>

Class Car

Protected:

+ string carName
+ int speed
+ int money
+ int tankSize
+ int totalGas
+ int availGas
+ int availCargo
+ int totalCargo
+ int availPassengerSeats
+ int totalPassengerSeats
+ Car::size carSize
+ Car::specialAbility special
+ Region pointer to location
+ Quest card
+ vector <Item> cargoItems

Public:

- Enum Car::size {small, medium, large}
- Enum Car::specialAbility{<list of special abilities for Car>}
- Get and Set Methods for string carName
- Get and Set Methods for speed
- Get and Set Methods for money
- Get and Set Methods for tankSize
- Get and set methods for totalGas
- Get and set methods for availGas
- Get and set methods for availCargo
- Get and set methods for totalCargo
- Get and set methods for availPassengerSeats
- Get and set methods for totalPassengerSeats
- Get and set methods for carSize
- Get and set methods for special
- Get and set methods for Quest card
- Method for addingItem

- Method for subtractItem
- Method for rolling Random Dice in Danger Scenario

```
// Below are child classes for Car class.  These will set attributes for each attribute within constructor
Class RallyCar : public Car
        RallyCar()
        ~RallyCAr()

Class LightCycle : public Car
        LightCycle()
        ~LightCycle()

Class VanDango : public Car
        VanDango()
        ~VanDango()

Class Mach5 : public Car
        Mach5()
        ~Mach5()

Class Airship : public Car
        Airship()
        ~Airship()

Class LoRyda : public Car
        LoRyda()
        ~LoRyda()

Class MtnCAT : public Car
        MtnCAT()
        ~MtnCAT()

Class Beaver : public Car
        Beaver()
        ~Beaver()

Class SolarSailor : public Car
        SolarSailor()
        ~SolarSailor()
```

<u>Region Header and Implementation File</u>

```
Class Region
Protected:
```

+ string regName
+ int difficulty
+ enum type

Public:

- Enum Region::type {<list of all regions>}
- Get and Set Methods for regName
- Get and Set Methods for difficulty
- Get and set methods for type

// Below are child classes for Region class. These will set attributes for each attribute within constructor

Class Mountain : Public Region
Mountain()
~Mountain()
dangerScenarioMtn(Car &)

Class Desert : Public Region
Desert()
~Desert()
dangerScenarioDesert(Car &)

Class Swamp : Public Region
Swamp()
~Swamp()
dangerScenarioSwamp(Car &)

Class Sky : Public Region
Sky()
~Sky()
dangerScenarioSky(Car &)

Class Urban : Public Region
Urban()
~Urban()
dangerScenarioUrban(Car &)

Class Cavern : Public Region
Cavern()
~Cavern()
dangerSenarioCavern(Car &)

Store <u>header and implementation files</u>

Class Store

Protected:

+ string objName
+ int cost

Public:

- Set and get methods for objName
- Set and get methods for cost
- Method for shopping in the store

Class Wrench : Public Store

Class Nas : Public Store

Class Ninja : Public Store

Class TracTires : Public Store

Class Gas : Public Store

Class Letter : Public Store

Class Food : Public Store

Class Water : Public Store

Class Traveler : Public Store

Class Trap : Public Store

<u>Quest header and Implementation files</u>

Class Quest

Protected:

+ string dialog
+ int ID
+ int questPass
+ int questItemNum
+ Store::Object object
+ Store::Object objReward
+ Car::special specialReward
+ int rewardMoney;

Public:

- Quest::card randomCard();
- Int roll dice

## Reflection

**What did you learn about the problem as you went? Why or how did you learn it?**

This is the final project of the course and the goals of the project were to satisfy the requirements, use inheritance, and have good OOP style. The goals equated to a culmination to everything learned throughout the term. In the last couple weeks, there was some learnings on the STL libraries and recursion. These I assume could be used in the final, but not necessary for the project.

Probably the thing that I learned about the project, was how to scale a large project such as this. Be able to design a piece at a time and imagine how those pieces will work with others. My initial thoughts going into designing the project was just how massive the scale was going to be compared to the previous projects I've worked on and also the limited time I was given. I decided to make the design somewhat high-level, leaving some areas to be able to fill in the details later as I move along in the project.

**What tests didn't work out the way you expected? What alterations did you have to make to your program due to failed tests? How could your planned tests have been more complete?**

I had some issues, putting together the circular queue of the track. Basically, what it was to be was a start point at the starting and finishing line location and then loop back to that same location. So instead, I created a way where when the race was going to begin you just start at the first location (in my mind, the location after the start/finish line) the proceed till you get to the finish location.

I had another issue with putting together the deque of quest cards. Implementing a way to randomly generate cards and then place each one into the stack. I decided, I would just 'roll dice' for a random number, and then whatever number it was would go to which type quest card would be created.

The tests for this project could have been more complete with being able to test out the finer details of the program. I did some of this testing without the use of the tests above, to save time. But it would be good to have these details in there to validate the correct data flow is being done.

**What was missing or needed to be altered from your initial design, and why?**

As I had mentioned above, I had some issues with the race track with a circular track. So, I instead modified the from the design where I would have the racer start at the first location after the start line, because nothing was going to happen in the game between the start line and first entering the first location.

Also mentioned above was the way in which randomly generating the quest stack of cards. The original solution was just make up a stack of cards, but this might be I could create a handful of them or a lot of script writing to make each individual card. Also, these would be in the same order during each game play. So the solution was to randomly make cards and sometimes there would be multiple of the same in the stack. The reason this was the case, is because I thought about this problem about leading up to creating the code.

**What problems did you encounter during implementation? How were you able to solve those problems? What outside sources (sites, books, or other materials) did you find helpful?**

I did have some problems during implementation, but that is only because I didn't quite remember how I accomplished the same thing in past assignments. I mainly used the course text book, with some help from online sources out on the internet such as cplusplus.com. For example, I had momentarily forgotten how to do inheritance, so I looked it up online because it was faster than going through my past assignments for assistance.

Another interesting problem that I ran into, was I ran into a problem that I later found out called, circular dependency. I had a few header files and with them had their own parent classes. Some of these classes included some other classes. Also, there were classes that were integrated with other classes. I had learned this could be a problem for the compiler. For instance, I believe it wasn't able to compile because if found undefined classes that it didn't know what to do with, but those definitions were later. I found a trick online to give it a brief definition, so that I could proceed with compiling.

**Can you generalize any parts of your problem solving experience in a way that might help you on future assignments?**

Start early, is the best way I can generalize the experience in solving the problems in this project. With the scale of this project and deadline, I probably could have started earlier on this project. But if I started later, I probably would have been rushed to complete the design phase and that would have caused greater problems in the overall project. If I start early, I can give adequate time toward each problem and to solve each of them through design.

A good message to use is to start early and setup a great design for the program. That way, the rest of the way into the implementation and testing will give less headaches. Also, it will give a less negative feeling toward the project.