Ryan DiRezze
CS 162 400
Final Project: Monster Hunting Game
December 4, 2018

FINAL PROJECT

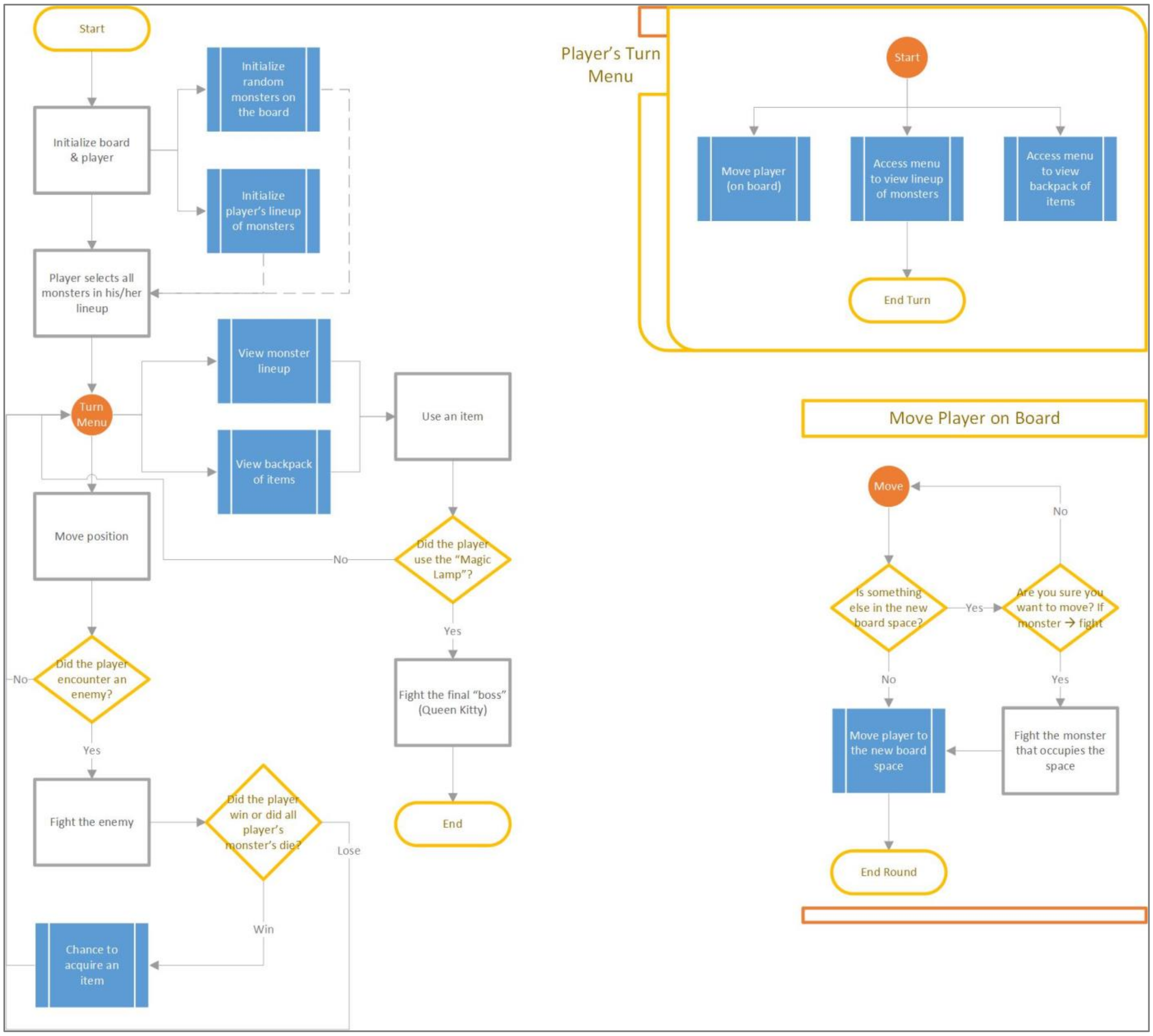
(DESIGN + REFLECTION)

Final Project (Monster Hunting Game) design and reflection

TABLE OF CONTENTS

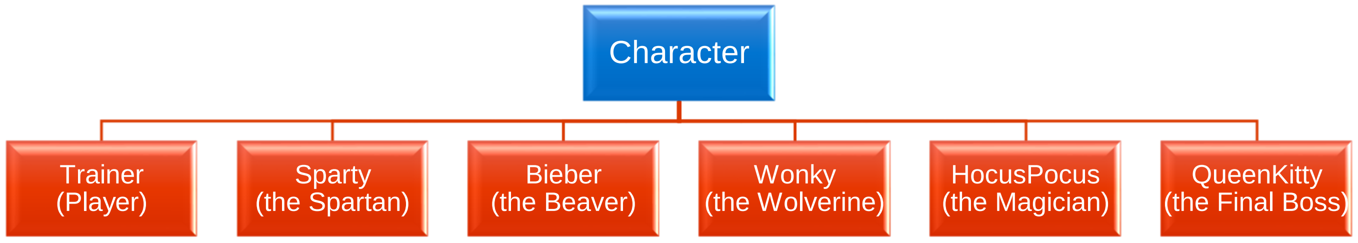
DESIGN / FLOWCHART

The below flowchart (on the left) illustrates the high-level flow of this program, including a more detailed flowchart (on the right) to illustrate a round of combat. A copy of the below flowcharts is also included within the assignment .zip file.


Start
Initialize board & player
Initialize random monsters on the board
Initialize player's lineup of monsters
Player selects all monsters in his/her lineup
Turn Menu
View monster lineup
View backpack of items
Use an item
Move position
Did the player use the "Magic Lamp"?
No
Yes
Did the player encounter an enemy?
No
Yes
Fight the final "boss" (Queen Kitty)
Fight the enemy
Did the player win or did all player's monster's die?
Lose
Win
End
Chance to acquire an item
Player's Turn Menu
Start
Move player (on board)
Access menu to view lineup of monsters
Access menu to view backpack of items
End Turn
Move Player on Board
Move
No
Is something else in the new board space?
Yes
Are you sure you want to move? If monster → fight
No
Yes
Move player to the new board space
Fight the monster that occupies the space
End Round

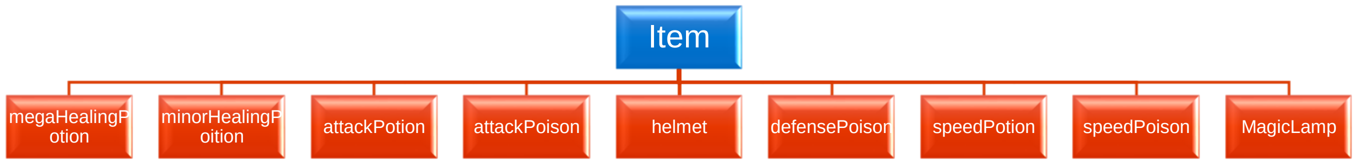
CLASS HIERARCHY

The below diagram illustrates the use of inheritance within this program for the user's character, which is *Trainer*, and the other monsters that are present within the game for combat. As shown below, *Trainer, Sparty, Bieber, Wonky, HocusPocus, and QueenKitty* classes inherit most, or all, of their member variables and functions from the base class, *Character*.



DATA STRUCTURE HIERARCHY

The below diagram illustrates the use of inheritance of data structures within this program for item objects, which the user, or player, finds throughout the game, including poisons and potions. As shown below, *megaHealingPotion, minorHealingPotion, attackPotion, attackPoison, helmet, defensePoison, speedPotion, speedPoison, and MagicLamp* all inherit their member variables from the *Item* data structure.



REFLECTION

This assignment reflection will cover the following content: (1) design changes while developing and implementing the assignment's program, (2) problems that were experienced and how those problems were overcome, and (3) what was learned from this assignment.

The final project was by far the most fun and entertaining assignment throughout this course, CS 162. The ability to interpret very loose requirements in a variety of ways, which allowed the developer to use his or her preferences, let me truly have fun with the assignment by using my imagination to develop a program design and implement that design. I would highly recommend that future classes continue to do this assignment; however, I would also encourage the instructors to give more assignments with, "loose requirements."

The reason why I would recommend that instructors give more assignments with, "loose requirements," is because the fact that the requirements were very, "loose," allowed me to have fun, try new things, integrate various concepts from the course, and challenge myself by trying related, but new things. For example, I recognized that I wanted to incorporate items into the game, which the player could find and use, so I had to decide on a method to incorporate these objects into my program. Although our class has opted to use classes when using inheritance, I chose to utilize data structures when defining the base and inherited data structures, which represent the, "items," that players find while playing the game. My decision to make this change was because I knew that my, "items," would only have a few simple member variables of string and integer type. With very simple member variables, the program didn't have to use any moderately complex functions, and I found that I am happy that I chose to use data structures for my game's, "item," objects rather than classes.

Although I am happy with my decision to use data structures within my program's, "item," objects, I experienced some trouble with implementation. Since this course has focused on classes with respect to inheritance and polymorphism, I wasn't quite sure *what* the implementation would look like for data structures with inheritance. The major challenge that I faced was with printing one of the member variables within my various, "item," objects. When I would try to print the member variables, I was unsuccessfully, as my program would only print blank content, even though a memory address would print when de-referencing the associated pointers.

Since the program pointed memory addresses, I knew that my objects were being created and used correctly; however, I wasn't sure why the string member variables would appear as blank content. What I found was that I had to create a default constructor for each data structure, including both the base and inherited data structures. With default constructors within each data structure, I was able to update the member variables of the *base* data structure, using inheritance, and I found that the reason no text would print, displaying blank outputs for the associated objects, was because my *base* data structure had a blank string, "", assigned to its string classes, and when I would try to print, the program would reference the *base* class' member variable values instead of my *derived* data structures. By creating default constructors for each data structure, I was able to update, and overwrite, the member variable contents of the *base*, "item," data structure, allowing the program to print the correct values on the screen.

Since this was the last assignment with fairly “loose,” but challenging, requirements, I decided to build just about all contents that I wanted to incorporate from my initial conceptual thinking. The example I just described was just one example of how I tried something related, but new, or unfamiliar, in order to implement a concept that I thought would be valuable to my program. With such a large, and complex, program, the above example was just one of many challenges that I faced when implementing this program. Although this assignment was one of the most challenging assignments during the term, I found this assignment to be the most *fun*.