Connie McClung
OSU ID#: 932990493
ONID email: mcclungc@oregonstate.edu
CS162, Section 400, Winter 2017

Final Project Design Document

Problem Statement

Create a text-based game for one player. Player will move through a series of spaces, gathering objects, and must achieve some objective. **Create an abstract base class for a space, and derive child space classes from it. Game must use at least 6 spaces of at least 3 different derived class types. Each space class must have at least 4 pointer variables.** Use the pointers to link spaces together to create a structure through which player will move. Each space should have a special action, such as a light that must be turned on, or controls.

The game must have an objective for the player to achieve. The goal can be random or change each time the player plays. The player must discover the solution to the situation/problem in order to win the game. **As the player moves through the spaces, there must be a way to keep track of which space the player is in. The player will carry a container with some limitation for capacity. The player will add items to the container that will be needed to achieve the game objective. The game should be organized so that player can move without requiring free-form input. The game needs a time limit (not necessarily an actual timer) to keep it from going on indefinitely. There must be a menu to allow the user to play or quit the game.** The objective of the game should be revealed in some way at the beginning of the game.

The game should have an interesting theme, but all other details are up to the programmer.

My Game Concept

I haven't played many text-based games, but the first thing that came to mind when I thought about moving through rooms was a haunted mansion theme. I got the idea of making a gothic mansion where a heroine learns her fiancé has been abducted and needs to be rescued.

I thought just collecting clues and objects could be boring, so I wanted to add a battle element like the one in our tournament project. I decided to make the heroine and all her enemies creature objects who could fight each other. If the heroine wins a battle, she gets the room's object.

Some objects will be required to rescue her fiancé when she finds him. If the heroine goes through every room and gets every object, at some point she will run out of space and discard an object. Not all objects will be required objects for winning the game. Each room also has a clue, but the heroine can collect an unlimited number of clues (they are almost weightless ☺). My original thought was to have lots of ways for the heroine to be weakened as she progresses, but I had to simplify that approach. I decided that she could lose strength in battle, and if she took too long to defeat an enemy. I gave her a way to be revived once if she died.

To put in a time element, I decided to have the fiancé get injured when he was abducted. As the game progresses, he would lose blood and possibly die before she reaches him, and game would be over.

To act on the spaces, the heroine will need to light her candle to illuminate the room. This can be done for only one room if she takes a random chance of searching for a fireplace ember to light her candle. Or she can obtain a permanent lighted candle by visiting the kitchen and lighting her candle at the hearth. If a lighted candle is obtained, the room property of being lighted will be switched on, allowing her to search for objects and clues. If not, she will need to find a way to light the room. Other ways of enabling actions on the space will be bool variables indicating whether the room's object and clue have been obtained, which will allow different menu displays and messages based on status. The heroine will interact with the space by facing the enemy creature that attacks her and obtaining the clue that is contained in each room.

If the heroine goes through the rooms of the house and gets to the roof, she will discover her fiancé and learn the identity of his abductor. The villain is generated randomly from 5 different choices, with different objects required to defeat the different villains. If the heroine has the correct object, she will rescue her fiancé and win the game. If not, she can go back and collect more objects, but the villain may change the next time she reaches the roof.

As the game progresses she will learn from the clues that her fiancé is a horrible person, so after she rescues him, she will leave the gothic mansion rather than marry him. If her fiancé loses too much blood or the heroine loses too much strength, the game will be over and she will lose.

Classes

Room Abstract Class

This class will define a space with 4 directional pointers to right, left, upstairs, and downstairs. It will contain a description function, getter and setter functions, and 3 virtual functions that can be overridden by derived classes: a menu function, a search room for clues function, and a fight danger for objects function.

Child Classes Inheriting from Room Base Class – these classes will inherit attributes of parent class with specific values, and will implement overriding functions for room specific menu(), searchRoom(), and fightDanger() functions. The derived classes will be:

- Guestroom – heroine's bedroom and game starting location. The enemy is a Vampire Raven, the clue is the letter from the abductor, and the object is her fiancé's signet ring(required to win)
- Kitchen – heroine can light her candle permanently here. The enemy is a Devil Hound, the clue is burnt letter suggesting the fiancé's dead son was accidentally poisoned, and the object is a medicine box(not required to win)
- Cellar – the enemy is a Haunted Suit of Armor, the clue is a skeleton hand of someone her fiancé apparently buried alive, and the object is a dagger(required to win)
- Library – the enemy is a Shadowy Intruder, the clue is a letter from a business associated accusing the fiancé of fraud and murder, and the object is a flask(required to win)
- MasterSuite – the enemy is a Murderous Footman, the clue is doctor's letter suggesting someone is insane, and the object is a handkerchief with strange monogram(not required to win)
- Attic – the enemy is a Ghostly Lady, the clue is a lady's diary, and the object is a locket(required to win)

- Roof - this is where the final showdown with the villain, and the possible rescue take place. There is not a standard clue or object. The clue is the discovery of her fiancé, tied up and bleeding. The fightDanger() function randomly reveals the villain. If the heroine already has the correct object, she will win and the game will be over.

Being Abstract Class – this is an adaptation of the Creature abstract class we created for Project 3. I used it to create a heroine and enemies to battle with her. The class is abstract and has virtual attack and defense functions that can be overridden by derived classes.
Child Classes Inheriting from Being Base Class – inherit attributes of parent class with specific values, with some variations (I mostly took these from the specific creatures we made before, and modified member variables and special attack / defense functions.)

- Heroine – like the Harry Potter creature because I didn't want her to die easily. She has a special defense where she comes back to life if she dies in battle, and a special attack skill to knock out her opponent if she rolls a high enough attack score. (Not in the class implementation: I use the counter for her first life to let me revive her one time in the game if she is too weak at the end of a menu cycle.)
- Vampire Raven (special defense: vampire charm- 50% of time, attacks have no effect on this creature)
- Devil Hound – like a Barbarian with tweaked die sides and values
- Shadowy Intruder – like a Barbarian with tweaked die sides and values
- Haunted Suit of Armor – has a Medusa-like ability to weaken opponent (special attack: attack roll reduces opponent strength by half)
- Murderous Footman (Deranged Servant) – standard creature with randomly tweaked die sides and values
- Ghostly Lady – like Blue Men (number of defense die decreases as strength decreases)

LinkedList template class– I implemented the container as a linked list template so I could remove an object from any position. I wanted to get practice using a template class, so I adapted the one from our textbook and made it a list of strings. I created 3 containers – a reticule (old-fashioned lady's handbag), a discarded objects list, and a clues list. Only the reticule has a capacity limit, since it holds the objects required to win.

Data members:

- String value
  ListNode*next
  ListNode *previous
- Struct definition to construct a list node
- Pointers to front and back of list

Allows implementation of linked list containing nodes of string values, and a next pointer, and a previous pointer. Functions to allow adding nodes to back, and removing from any position. Class also contains a display queue and clear queue function.

Game Class

Contains the menu-based game loop in the play function, and an intro function. Contains variables for rooms, heroine, containers and various bool status variables, along with getter and setter functions.

I adapted the idea of the menu-based play loop from the example final project by Christopher Merrill, because it made the most sense to me for the game concept I had already come up with. I tried other implementations, but they really didn't work as well for navigating without text input for the space I wanted.

But I wrote the code in my own way, made numerous different implementation choices, and had a different idea of how the game progressed to win or loss. For example, I have all the special room functions as member functions of the room classes. I made the menu options more consistent throughout the game. I used switch structures for menu options, and have the room change submenu in the game class instead of duplicated in individual rooms. I implemented the container as a linked list template, and added a detailed submenu for viewing and removing objects and clues. I created a Heroine class for the player, and added creatures for the player to fight in each room. I added a separate room function for fighting danger before an object could be obtained. I added a random function allowing heroine to illuminate each space, and a way to weaken and revive the heroine to add some variety to her strength level. I used a timing element that was like the example project, but this option of decrementing or incrementing a counter after each round (in my case after menu loops) to add time urgency was also recommended by a TA on Piazza, and I tried to program mine differently. I didn't look at the full program for this example, but I think the way I resolved the objective to win is different as well.

I think my program is not as smooth or well-coded as the example, but it is my own implementation. I was worried about taking an idea from another person's work, but it seemed counter-productive to use a worse method for implementing the game loop to be different, so I worked hard to differentiate my code, while crediting the original project as inspiration.

Input

This is a menu-driven game. The input consists of integers to select menu options, and in one place requires user to input the name of an object to be removed from reticule. The integers must be validated.

Processing

Create game and instantiate variables. Run the game intro and play function, which calls Room child class functions and Being child class functions. The main play loop shows menu for current location, processes menu selection, and re-displays menu until location is changed. Game continues until loss of strength triggers game over, or until the heroine reaches the roof with the required items.

Output

Main menu to play or quit. User prompt for invalid data entry. Display game intro text. Menu prompt for each room. During heroine-enemy match-ups, display name, creature type and starting strength of each opponent. Show effect of attack and defense rolls on strength score for each round. When a creature's special attribute affects scoring, display notice on screen Display result of each battle. Display clue, object and discarded object name, description, and status. Display villain backstory and game win/loss status. Validate all integer inputs for menu selections.

Design Change Log

When I wrote my original idea down, I wrote out a more complicated, meandering journey for the heroine with more ways for her to lose and add strength. But to do this game without free-form input and without being too boring, I decided it worked better to have consistent menu options throughout and add a little excitement by putting in battles and interesting clues. This decreases the game difficulty, but was simpler to implement consistently. I stayed with my original idea of the rooms, the enemies, impact on heroine's health, and the idea of randomly determining the villain identity.

I had to simplify the way the winning solution is achieved. I was having trouble with how I determined if the heroine had the correct object for the win and the logistics of trying to retrieve previously discarded necessary objects, so I prevented her from discarding any item that is required to win.

I started out implementing the containers as vectors of item objects, but it was too hard to display numbered lists of items and to delete from any position. So I looked over our text and saw that there were linked list examples allowing deletion from any position, not just front or back. I decided to use a template list class, and implement the containers as lists of strings instead of items. But I kept the item objects as member variables of each room child class, for ease of displaying detail about objects. The drawback of my container implementation is that the user must type in the name of the item to delete. I also think there may be an issue with clues displaying more than once in my clue list if the user navigates a certain way, but I did not have time to test enough to correct this bug in an optional section of the program, and am not sure if it still exists.

I had to make a lot of changes in the way I passed room status variables to trigger different menu displays. I went back and forth between setting them inside rooms versus setting them inside the game loop. I'm not sure I am completely happy with the way I finally dealt with this issue.

At first I had the enemy difficulty too low, then too high. I decided that I didn't want the heroine to die in battle very often and end the game prematurely, so I tweaked these properties. Right now, the heroine wins most of the time, but gets weak if battle goes on too long.

Reflections

I don't know much about text-based games. But I liked the concept/story I came up with, and am relatively happy with the way the game runs. As usual because I work full-time, I had time constraints, especially as an inner-ear disorder sent me to bed for 2 days last week.

I struggled with how to use the provided examples as inspiration without copying specific code. I would have been lost without these examples, since I have never played a text-based game, but I also felt anxious that modeling any of my program on them would be considered cheating. I don't have enough experience with these types of games to know what are standard techniques used in many games versus an original idea. So I tried really hard to make my implementation code individual, but I used a menu-based loop because it was the best solution for my game structure. It seemed stupid to avoid the best technique out of fear of duplicating someone else's approach. If I had it to do over again, I probably would try not to look at the examples at all until after I had written a complete program on my own, so I wouldn’t be paranoid about accidentally copying things. But that would have had its own drawbacks, as I might have come up with totally unworkable methods to implement a text game without freeform entry.

As usual, I had challenges freeing memory with linked lists. Even though I based my code on an example in our textbook, I still had memory leaks! I finally found where the list nodes were not being freed. I added my enemy objects to the room child classes just to make it easier to de-allocate them. I had a much easier time dealing with pointers to objects this project, because I made sure to use getter and setter functions. I barely have any difficulty de-referencing pointers now that I use this approach.

If I had more time, I would try to improve my container implementation, make the battle match-ups more exciting, and improve the interaction between the objects obtained and the game solution. I would also maybe add an option to fight the villain if the heroine doesn’t have the required object, and consider other ways to work a time element that are not counter-based. I would also do more functional decomposition; I think I duplicated more patches of code between room classes than I should have.

I would spend more time debugging, to be sure I ran through as many variations as possible. I did a lot of it, and found lots of little changes that I made, and both won, lost, had heroine die and be revived, had close battles and not close ones, etc. But I think debugging on something like this could go on much longer.

Final Test plan

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
Test menus for input validation, correct navigation; test that all menu paths work correctly and change as status variables change; test that room changes work correctly, test that container management menu works correctly	out of range selections. Change rooms for every room, in and out of ideal sequence, cycle through rooms and collect all objects, clues, then cycle through without collecting, then cycle through and try functions after already collected.	Room child classes menu(), fightDanger(), searchRoom() Game class play(),	Invalid integers not allowed, invalid string inputs for remove items not allowed, display changes if objects /clues are obtained, menu choices implement differently if room is lit or not lit, location description changes when player changes rooms, players can navigate through any sequence of connected rooms.	Invalid inputs rejected, rooms change correctly, display changes if room is lit, room lighted status changes if player has lighted candle, display changes if objects/clues are already obtained, player can navigate through all linked spaces correctly, container objects can be added, displayed, removed
Test bool and strength variable updating	Check that heroine gets weaker in certain situations, that she revives, that menus change with room is dark versus illuminated, that menus change when villain is defeated, that the lord's	Room functions, game play function, attack and defense functions for Being child classes.	If room is not illuminated, menu functionality decreased, if heroine gets lighted candle, bool variable for lit room changes, if heroine gets week enough she will die and come back to life once, if enemy points	Menus different when room lit or unlit, random candle lighting function works, obtaining lighted candle illuminates all future rooms visited, if heroine dies in battle, she revives, heroine usually defeats enemy because I set it that way.

	strength decreases to trigger time warnings, that the correct winner and win/loss status displays		decrease to <= 0 they will be defeated, if villain is defeated game will be over, if time element runs out, game will end	
Test display of text for typos, grammar, confusing language	Check that clues, text, spelling, make sense when you navigate through the game in different sequences	Game intro Item description, clue description and names, room child class menu, fightDanger(), and searchRoom()funcions	Minimal misspelling, sentences make sense, clues hint at possible villains, instructions make sense.	Mostly comprehensible, I am not sure if it is totally clear to someone other than me, the clues are not supposed to tell the whole story, the villain reveal makes sense on its own.
Test for memory leaks	Run full program with as many options used as possible and see if all memory is freed, run multiple times	all classes, main, constructors/destructors	all allocated freed	All heap blocks were freed, no leaks are possible. Valgrind gave two warnings about variables set but not used, but those are not memory leaks.