Final Project

Oliver Solorzano
CS 162
19 Mar 2019

Problem:

Need to design a program that meets the following conditions:

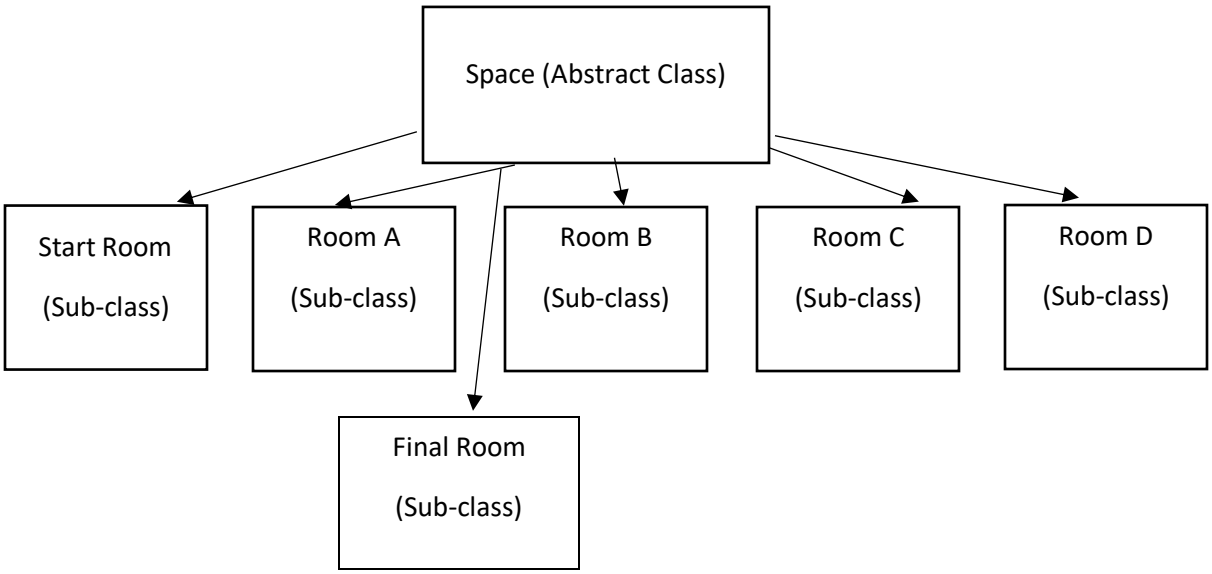
1. Has theme
2. Utilizes a “Space” Class
    a. Abstract
    b. Has virtual functions
    c. Uses 4 space pointers
        i. Top, bottom, left, right
    d. Unused pointers are set to NULL
    e. At least 3 derived classes
        i. At least 6 spaces
    f. Player can interact with “Space” structures
3. Keeps track of player
    a. Printable map
4. Player has container
    a. Holds items via linked list
    b. Any item in list can be accessed
5. Time limit
    a. HP limit
    b. Movement limit
6. Has set interface for player
    a. Introduction must give instructions and background to player
    b. The input is restricted
        i. Movement
        ii. Interaction
        iii. Inventory access
        iv. Map
7. Can be text  based gameplay or visual

Initial Design:

1. Future theme, 1984, George Orwell(ish)
    a. Correction facility
2. Space Class (Abstract)
    a. Child classes
        i. Starting room
        ii. Rooms A/B/C/D
        iii. Final room
    b. Has virtual functions
        i. Get/set pointers
        ii. Trials/challenges for each room
    c. Has 4 space pointers
        i. Top, bottom, left, right
        ii. Only uses left/right, one directional progress

    d. Unused pointers are set to NULL
    e. Player can interact with space structures
        i. Room trials affect player inventory/score
3. Game class
    a. Links rooms
    b. Controls game progression
4. Keeps track of player
    a. Prints location and next/previous room
5. Player has container for keycards
    a. Holds items via fixed array
    b. Max count is 6
6. Time limit
    a. Either completes, quits, or loses
7. Has set interface for player
    a. Introduction gives instructions and background to player
    b. Each scenario has menu for player
        i. Decision
        ii. Interaction
        iii. Inventory access
        iv. Quit

Class Hierarchy:



Final Project

Oliver Solorzano
CS 162
19 Mar 2019

Test Table:

| Test Case | Input | Expected | Observed |
|---|---|---|---|
| User enters option to quit | Enter int associated with quit option | Program exits | Program exited |
| User enters invalid menu selection | Enters value != int, or not within range | Tells user entry is invalid, tries again | Tells user entry is invalid, tries again |
| Game resets when user score hits zero | Play game, fail tasks until score reaches zero | Game displays losing message, resets | Game displayed losing message, reset |
| Game adds keycards to inventory appropriately | Complete tasks one by one, print inventory | Keycards show up in inventory after completing task | Keycards showed up in inventory after completing task |
| Player can't proceed without right keycard | Try to pass through each door without right keycard | Needs right keycard for every door | Needed right keycard for every door |
| After answering correctly, can no longer answer again | Complete each task, get keycard, try to complete again | Won't be able to complete task more than once | Wasn't able to complete task more than once |
| Game ends when player reaches final room | Reach final room | Game displays winning message, ends | Game displayed wining message, ended |

Reflection:

Of all the projects in this course, this was the hardest to structure. Usually we're given some sort of framework and we fill in the gaps. This presented new challenges and different ways of thinking about design. I kept my design simple, mainly because I wanted to focus on functionality. I set out to create a text-based game with menus and scenarios unique to each room. The complication was coding the program to ensure interaction between the player and space structures. After some difficulty I figured that I could pass in the player object by reference into each room, and let the different rooms affect the object directly. This was mainly in the form of adding items to inventory, checking to see if the appropriate keycard was in the inventory, and altering the score for each failed task. Besides that, there was not much coding difficulty. All the techniques I employed in this program had been used at one time or another in this class so I was well aware of how to properly implement them.