Final Project Design + Reflection

Problem

This is the final project and you will be able to design and implement a one-player, text-based game where the player can move through spaces to get items and accomplish goals.

You have the freedom to decide what kind of them you would like to create for your game as long the requirements below are met.

General Design

I'm not one to bite off more than I can chew. I'm sticking very close to the requirements. The theme of my game is a Haunted House. There will be 6 rooms total with 3 different kinds of rooms: room, candle room, and ghost room. The object of the game is to get the candle, bring it to the ghost, and exorcise the ghost. While traversing through the house, the user will gain or lose courage which will affect gameplay. For example, the user needs at least 50 courage to get the candle, and at least 75 courage to exorcise the ghost. If the user goes to the ghost with too little courage or without the candle, it is an immediate game over. There will also be a limitation of 10 moves to achieve the objective.

Input Validation Design:

I will recycle my InputValidation class used in Project 1. It has an overloaded function called getInt(). I chose to make InputValidation its own class so that it could be used in future programs. getInt() receives input from the user, converts it into a stringstream, which then checks to see if there is anything beyond an integer that is left over. If there is, it makes the user try to input an integer again.
getInt(int min, int max) serves the same function as getInt() but also runs another validation checking input to see if it lies between min and max. getInt(min, max) will be useful for specifying minimum and maximum values for the following inputs:

- Menu options for moving left, up, right, or down

There is also an InputValidation function called pressEnter that will help pause the game at certain moments to allow the user to process information. After the user pushes enter, the game will continue.

Menu Design:

I will recycle my Menu class used in Project 1 to display menu options, get input from the user, and store that input as a user choice variable. I chose to make Menu its own class so that it could be used in future programs. It references my Input Validation class to ensure the user is entering a valid input from the menu. There are functions to add an option to the menu, print the menu options, ask the user for his/her choice, and return the user choice.

The only menu included in my game will be a move menu that lists four directions to move for the user to select: up, right, down, left.

Space Class Requirements:

1. The game requires a Space class, which represents the space the player can be in. The Space class must be an abstract class that will have pure virtual functions.

Inside the Space class, there must be at least 4 Space pointers: top, right, left, and bottom.

Use the class to create a game with the structure of linked space. (You are free to add more Space pointers to the Space class, but must have at least 4 Space pointers)

Note:

- Even if your structure is linear, such as a train, you still have at least 4 pointers in the Space class
- Any unused pointers need to point to NULL.

1. The game must have at least 3 derived classes that are derived from the Space

Each representing a different type of space, and need to have a special action for the player to interact with. It can be opening the door to another space, or maybe attack the monster, or turn on the light switch, or sing a song to please the king.

- The game must have at least 6 spaces.

Space/Room/GhostRoom/CandleRoom Class Design:

Space class will be an abstract base class representing room spaces in a Haunted House. Space has the following derivative classes: Room, GhostRoom, and CandleRoom. Room will be a basic space that adds or subtracts between 0 and 25 courage to/from the user's courage variable. CandleRoom gives the user a candle to exorcise the ghost so long as the user has 50 or more courage. GhostRoom contains the win condition for the game so long as the user has 75 or more courage and the candle. Should the user visit the GhostRoom without the candle or enough courage, the user loses the game.

Gameplay Requirements

1. The game must have a theme. It can be a crime-solving theme, a fantasy game. The game must also have a goal for the player, so the player can complete the goal to achieve victory.

2. The game must keep track of which space the player is in. It can be in a form of visualized space, by printing the map out and indicate where the player is, or printing text describing where the player is at and what adjacent space is around the player's space.
3. You must create a container for the player, to carry "items". The container must have a capacity limit. The game must contain some items for the player to obtain in the game and one or more of these items must be required as part of the solution to accomplish game's goal, such as a key to open a locked door, etc.
4. The game must have a time limit, which limits the amount of time/steps/turn the user can take before losing the game. The following are some notable examples of the time limit:
    1. Step limit that limits the number of times the user can switch spaces.
    2. Health system which decreases the player's health point from space to space, and maybe some painkillers scattered around the spaces.

Note: make sure you give enough steps to allow the game to perform testing.

5. Again, the user must be able to interact with parts of the space structure, not just collecting items to complete the game.

Interface

1. At the beginning of the game, the goal of the game must be declared and printed to let the player know the goal of the game.
2. The game cannot contain free-form input. An example of free-form input would be to type out "kitchen" to go to the kitchen space in the game. It is tedious and counter-intuitive.
3. The game must provide the user a menu option for each scenario of the game.
4. You are not required to have a printed map to visualize space, a text-based game is sufficient. But, it would be great to have a printed map, it is easier to interact with, and it's cool to show a cool game with a map to friends and family.

Game Class Design:

Game is the main engine for the flow of the Haunted House game. It creates the haunted house as a linked list-like object. It also prints the house each turn and controls the gameplay for each turn. It keeps track of the user having a candle with a bool variable. It also keeps track of the user's courage. Most of what runs the game will be found in the Game class. There will be a playGame function that controls the flow of how the game plays.

Main Design:

main will not contain much other than seeding random numbers, creating a game object and calling the playGame function.

Reflection

I approached this program methodically starting with importing the easy stuff: my already written Menu and InputValidation classes. I find that starting out with these small victories gives me confidence when the project looks daunting at first.

I can honestly say I experienced NO problems when coding this game. I think the hardest part was coming up with the concept. As previously mentioned, I was not about to commit to anything that I couldn't finish in time and therefore kept it simple. I'm not sure if I set the bar too low for myself or if I'm just becoming a better programmer where stuff I used to find difficult is no longer. In either case, it was nice to finish with time to spare and a decent outcome. I know that my game won't be the best of all created, but I enjoyed making it and playing it.

Test Tables

For InputValidation Class **–** This class was already tested in Project 1

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes
input too low or high	input < 2 input > 200	getInt(2, 200)	You must select an integer equal to or between 2 and 200.	You must select an integer equal to or between 2 and 200.
input in correct range	2 < input < 200	getInt(2, 200)	Returns input	Returns input
input extreme low	input = 2	getInt(2, 200)	Returns 2	Returns 2
input extreme high	input = 200	getInt(2, 200)	Returns 200	Returns 200
input not an integer	input = a	getInt(2, 200)	You must enter an integer value.	You must enter an integer value.
input not an integer	input = 123abc	getInt(2, 200)	You must enter an integer value.	You must enter an integer value.
input not an integer	input = enter key	getInt(2, 200)	You must enter an integer value.	You must enter an integer value.
input not an integer	input = asdf 123	getInt(2, 200)	You must enter an integer value.	You must enter an integer value.

For Menu Class - This class was already tested in Project 1

Test Case	Input Values	Driver Functions	Expected Outcomes	Observed Outcomes

input option string	"Start Langton\'s Ant simulation"	addOption(string)
		printOptions()
		main()
			1. Start Langton's Ant simulation	1. Start Langton's Ant simulation

input option string	"Quit"	addOption(string)
		printOptions()
		main()
			1. Start Langton's Ant simulation
			2. Quit
				1. Start Langton's Ant simulation
				2. Quit

input too low or high	input < 1
	input > 2
		getInt(1, 2)
		promptUserChoice()
		main()
			You must select an integer equal to or between 1 and 2.	You must select an integer equal to or between 1 and 2.

input not an integer	input = a	getInt(1, 2)
		promptUserChoice()
		main()
			You must enter an integer value.	You must enter an integer value.

input valid	input = 1
	input = 2
		getInt(1, 2)
		promptUserChoice()
		getUserChoice();
			returns 1
			returns 2
				returns 1
				returns 2