Chermaine Cheang

Final Project Reflection

Originally, I was planning to have my class hierarchy as follow, Office, Rooftop, and Lobby will inherit from abstract class Space, Conference Room and Break Room will inherit from Office, Basement will inherit from Rooftop, and Cafeteria inherit from Lobby. However, because I have my constructor for each class to call on the function to create and add objects into each room every time the constructor is called, my original plan to inherit some of the class from their supposed parent class (Office, Rooftop, and Lobby) failed because the derived class will always called on the parent's class create object functions. This caused the derived class to have objects not needed in their container to hold objects. As a result, I had all the class inherit from the abstract class Space directly.

In addition, I also made some changes to Lobby, Cafeteria, Kitchen, Basement and Rooftop classes to handle the requirement to add and delete a node during run time. To illustrate, I added data member variables to keep track of whether a space, specifically Kitchen, Basement and Rooftop, has been built in Cafeteria and Lobby classes. These were to make sure that I only added these additional nodes to my linked structure (floor plan) once. Another data member I added was the variable to keep track of whether a person is rescued from the space. Since, the game consists of three people where the player will have to save, these three people will each be in Kitchen, Basement and Rooftop, and once the player has saved a person from these spaces, the data variable will be set to true. This is to indicate that the player has completed his/her task in this space, hence, I can safely remove this space from the game. Moreover, I also added two pointer variables in Lobby class to hold the address of Basement and Rooftop object when these rooms were built.

Some changes I made in the Object class were to exclude all mutator functions for private data members because these values do not change once an object is created. Originally, I had a data member to keep track of where the object will be after it is created, however, since all objects were defined in their respective space and no reuse of objects throughout the program, I removed this data member completely from Object class. I added a data variable, interacted, to keep track of whether the player has interacted with an object. This was mainly to prevent player from collecting name tags from the same location. Therefore, every time the player found a name tag from an object, the interacted variable for this object will be set to true, so no further interaction with this object will be possible. Another major changes I made was changing the queue container from holding an object to holding a pointer to an object. This were to facilitate the interaction of player with each object in the space, to make sure that the player interacts with the same object every time they are in the same room, and to keep modification of each object consistent throughout the program.

In addition, previously I had my getObjectInSpace() function to be a constant, thinking that it is an accessor function. However, when I tried to compile the program after implementing the Object class, my program ran into error. Upon investigation, I realized that this function caused the error due to the changing of the queue container inside this function. That is elements in the container were being removed using the pop() function, thereby changing the elements in the container; hence, this function, although it is an accessor function, cannot be constant because elements in the container were not constant. Another error that I ran into while testing at the beginning stage was running into segmentation fault every time I tried to access objects in a space. This error was due to the fact that I did not call the function to create object for each space in their constructor. Hence, every time I ran the program, I was trying to access objects that do not exist. To solve this problem, I edited all my class's constructors to call createObject() every time the constructor is called.

Originally, I planned to generate the order of name tag in main. However, after much coding, I realized it will be easier to have these functions for generating and creating names specifically in the Character class. Again, this function will be called by the constructor when a character is being created. Another roadblock I encountered was after generating the order, the createName function failed to push names into the stack container. After much debugging, I realized that values passed from randNameOrder into createName differed when I attempted to use those values in a for loop. As a result, I changed randNameOrder to a void function instead of the previous int* function, and have randNameOrder call createName function passing in the three integers created in their order without using any array or pointer. I also changed createName to a void function from its original string* because I can access the array holding the name directly from the function since all these were in the same class. After these changes, I was able to push values into the stack container successfully. Since i have an array for keeping track of the names in which the player will rescue, my solutionMenu() and printSolution() functions were changed to a more simpler version by using the array stored in each character, instead of having to pop an element from the stack and push it back once all elements are popped.

Additional changes made within Character class was replacing vector container to hold items in pocket to a queue container. This was because of the difficulty in accessing element within the vector. Once items were added to the element, I had to locate the item by iterating through the entire vector to find the item I want. After locating the position of the item needed, my original plan was to use the built-in erase function to remove element at this position. However, this caused an error. I tried first swapping the element with the last element in the vector before deleting the element from the container, but this approach failed as well. As a result, I decided to use the queue container which I was more familiar with. Since one of the requirement stated that I have to limit the number of elements the queue container can

hold, I had the setItemInPocket function return a bool value to indicate if item was added to the container successfully. A function, checkIfItemInPocket, is also added to check if the item the player is looking for is in the queue container holding all the items the player has collected. If the item is not in the container, the function will return false. If the function is in the container, then the function will return true. This is to make sure that no illegal popping of element from the queue container is to be performed. I also added a data member in Character class to keep track of the number of person the player has to saved in order to win the game.

In addition, my original design was to set pointers connecting spaces in createSpace by calling the mutator functions for each pointers individually. However, in order to reduce the number of lines of code and made the function simpler, I decided to add a function which will take five pointer to space as parameters. These five pointers will then be used to link all the spaces to one another according to the floor plan. One major changes to the floor plan was that I used all the four pointers in each space instead of the original plan of having some rooms with only three valid doors connecting to other rooms. The main reason for this changes was to keep track of where the player wants to go easily. The original plan with different number of doors for each room made it hard to move player to the next destination. With this changes, I was able to safely assume that when player selected '1' from the list provided, the player wished to move to the room on the North side of current room.

To take into consideration the fact that some rooms might have more than four pointers, additional if statements in doorsToNextSpace, the function that print out list of rooms connected to current room, were added. These if statements checked if current room is either Cafeteria or Lobby, and provide additional list to if the pointers to those additional rooms are valid. However, I soon ran into another segmentation fault when trying to delete Kitchen space after the player has saved someone in the kitchen. This was due to the accessing of deleted information in doorsToNextSpace. To resolve this issue, I check if pointers pointing to Kitchen has a null value or not before listing Kitchen as an option in the list.

Freeing dynamically allocated memory for this program was a hassle and took the longest to complete and fix. Firstly, I added a freeMemory function which will check if all the four pointers in current space is pointing to a valid space. If there are, then I will call to delete these spaces before deleting current space. However, adding this function still did not solve the problem of memory leaked in my program. Then, I realized I had to free memory allocated to objects in each before deleting the space because these objects were also dynamically allocated. Hence, I added another function, deleteObject, which uses a while loop to delete object in each space. Although adding this function was able to free more memory, I still have the some unfreed memory at the end of the program. Upon much investigation and debugging, I realized that not all objects in each space were deleted before the space was deleted. As a

result, I added a checkObjectInSpace function in abstract Space class to check whether the queue container holding objects in each room is empty before calling the queue popping function in getObjectInSpace. This successfully free all memory allocated for the initial spaces created at the beginning of the game and all objects in these rooms, but not those memory allocated for additional rooms added throughout the game. I kept getting invalid read when the system is trying to delete the three additional spaces added. After debugging for a very long time, I realized the problem is solved when I delete these spaces in separate individual functions instead of having them all in one function.

Testing Plan

Test Plan	Input Value	Drive function	Expected outcome	Test Result
1	Prompt user to start the game	startOption()	Start menu is displayed	System displayed: *Start the game?* *Y or N* *(wait for user input)*
2	Validate user's input value from startOption() i. user enters Y ii. user enters N iii. User enters J	yesNoValidate()	Function returned i. True ii. True iii. False.	Function returned i. True ii. True iii. False. System displayed: *Invalid input.* And prompt user for input again.
3	Create an object	object()	Object is created with parameters provided.	Object is created, and object's name and description is printed out correctly.
4	Create an Office	Office()	Office is created and information is printed	An Office object is created successfully and information is printed out correctly.

5	Create a Lobby	Lobby()	Lobby is created and information is printed	A Lobby object is created successfully and information is printed out correctly.
6	Create a Cafeteria	Cafeteria()	Cafeteria is created and information is printed	A Cafeteria object is created successfully and information is printed out correctly.
7	Create a break room	BreakRoom()	Break Room is created and information is printed	A Break Room object is created successfully and information is printed out correctly
8	Create a Conference Room	ConferenceRoom()	Conference Room is created and information is printed	A Conference Room object is created successfully and information is printed out correctly
9	Create a Kitchen	Kitchen()	Kitchen is created and information is printed	A Kitchen object is created successfully and information is printed out correctly
10	Create a Basement	Basement()	Basement is created and information is printed	A Basement object is created successfully and information is printed out correctly
11	Create a Rooftop	Rooftop()	Rooftop is created and information is printed	A Rooftop object is created successfully and information is printed out correctly
12	Create all the rooms needed to start the	createSpace()	Successfully created Office, Lobby, Cafeteria, Break Room and	Office, Lobby, Cafeteria, Break Room and Conference Room are created.

	game		Conference Room.	
13	Link Office to: Break Room, Conference Room and Cafeteria	createSpace() setDoorToS() setDoorToE() setDoorToW()	Linked successfully.	Office is linked to Break Room through the East Door, Conference Room through the West Door and Cafeteria through the South Door. All links are correct.
14	Link Cafeteria to: Office, Lobby, Break Room, Conference Room	createSpace() setDoorToN() setDoorToS() setDoorToE() setDoorToW()	All links are linked successfully	Cafeteria is connected to the Office on the North, Lobby on the South, Break Room on the East, and Conference Room on the West. All links are correct.
15	Link Lobby to: Cafeteria, Break Room, Conference Room	createSpace() setDoorToN() setDoorToE() setDoorToW()	All links are linked successfully	Lobby is connected to the Cafeteria on the North, Break Room on the East, and Conference Room on the West. All links are correct.
16	Link Break Room to: Office, Lobby Cafeteria	createSpace() setDoorToN() setDoorToS() setDoorToW()	All links are linked successfully	Break Room is connected to the Office on the North, Lobby on the South, and Cafeteria on the West. All links are correct.
17	Link Conference Room to: Office, Lobby Cafeteria	createSpace() setDoorToN() setDoorToS() setDoorToE()	All links are linked successfully	Conference Room is connected to the Office on the North, Lobby on the South, Cafeteria on the East. All links are correct.

18	Prompt user for name, and create the character with name entered	Character()	Character created successfully with user entered name, and starting location is office.	System displayed: *Enter your name.* *Chermaine* *Welcome, Chermaine. You are currently in Office.* Character successfully created and initial location is correct.
19	Test if randNameOrder() generate three numbers (1, 2, 3) correctly.	randNameOrder()	Three different numbers in the range of 1-3 are generated.	System displayed the three numbers generated from randNameOrder(): *2 1 3* Which is a correct behavior.
20	Add names into Stack container holding nameTags	createName()	Names successfully added into Stack.	System displayed the last name added to the Stack, Sansa, followed by Jon and lastly Brandon, which is the correct behavior.
21	Prompt user if they want to know the solution	solutionMenu()	Menu displayed and waits for input from user.	System displayed: *Do you want to know the solution?* Wait for user's input.
22	User wants to know the solution	printSolution()	Print out the solution in the correct order.	System displayed: *Sansa's name tag is in ...* *Jon's name tag is in ...* *Brandon's name tag is in...*
23	Interact with objects in a	interactWithSpace() interactWithObject(	System displayed where the player	System displayed: ***Currently in Office***

	space	)	was and the first item in the container, and asked if user wants to search through this object. If user entered no, then user will be asked if they want to move on to another object. If user entered no, then they will be asked if they want to move on to another room. If user entered yes, then interaction with the object will take place, and any findings will be reported.	*Jon's Desk is right in front of you.* *Do you want to search through Jon's Desk?* *Y or N* ***Y*** *Description of object…* *Did not find anything.* *Move on to next object?* *Y or N* ***Y*** *Bookcase is right in front of you.* *Do you want to search through it?* *Y or N* ***N*** *Move on to next object?* *Y or N* ***N*** *Move on to another room?* *Y or N* ***Y*** System displayed a list of room connected to Office and waited for user input.
24	User found a name tag.	foundNameTag()	Informed player that they found a name tag.	System displayed: *You found Jon's name tag.*
25	Ask if user	keepNameTag()	Prompt user if	System displayed:

	wants to keep the name tag. User wants to keep Jon's name tag.		they want to keep the name tag. If user wants to, add name tag to container successfully.	*Do you want to keep Jon's name tag in your pocket? Y or N* ***Y*** Jon's name tag added to Queue container successfully.
26	User does not want to keep the name tag	foundNameTag() keepNameTag()	Name tag returned to Stack.	Pushed Jon's name tag back into Stack container successfully.
27	Ask if user want to move on to another object after keeping the name tag.	interactWithObject() moveToNextObject()	Prompt user if they want to move on to next object and wait for input.	System displayed: *Move on to next object? Y or N* Wait for user input.
28	Player wants to move on to next object.	setObjectInSpace()	Current object is added back to Queue, and next object is taken from Queue for next iteration.	Current object successfully added to the back of Queue container holding objects. First object in container is removed successfully. Next cycle of interaction continued.
29	User wants to move to Cafeteria	movePlayerToNext() addNewSpace()	Player is now in Cafeteria. Kitchen is built and one of Cafeteria's pointer is pointing to Kitchen.	System displayed: ***Currently in Cafeteria*** *...* *..* When exiting Cafeteria, system displayed: *This room is connected to 1 - Office*

				2 - Lobby *3 - Break Room* *4 - Conference Room* *5 - Kitchen* This indicated that Kitchen space was added and linked successfully.
30	User enter kitchen and saved Jon.	foundBody() deleteKitchen()	Player saved Jon and upon exiting Kitchen, the space is deleted	System displayed: *Good job! You saved Jon.* *Move on to next space?* *Y or N* ***Y*** *Deleted objects in kitchen* *Deleted Kitchen* ***Currently in Cafeteria*** This indicated that memory allocated for Kitchen was freed successfully.
31	User enters Lobby	movePlayerToNext() addNewSpace()	Built basement and rooftop successfully and linked to Lobby.	System displayed: ***Currently in Lobby*** *...* *..* When exiting Lobby, system displayed: *This room is connected to* *1 - Cafeteria* *2 - Office* *3 - Break Room* *4 - Conference Room* *5 - Basement*

				6 - Rooftop This indicated that Basement and Rooftop spaces was added and linked successfully.
32	User enter Basement, but did not saved the person.	movePlayerToNext()	User exited basement successfully, and when prompt asking which room to enter from Lobby, Basement is still an option, because it is not deleted yet.	System displayed: ***Currently in Basement*** *...* *You do not have Jon's name tag.* *Come back when you have it.* ***Currently in Lobby*** *...* *This room is connected to* *1 - Cafeteria* *2 - Office* *3 - Break Room* *4 - Conference Room* *5 - Basement* *6 - Rooftop*
33	User enter basement, saved Jon, and exited Basement.	movePlayerToNext() deleteBasement()	Successfully saved a person in Basement, and deleted Basement when player exits.	System displayed: *Good job! You saved Jon.* *Move on to next space?* *Y or N* ***Y*** *Deleted objects in Basement* *Deleted Basement* ***Currently in Lobby***

				This indicated that memory allocated for Basement was freed successfully.
34	Played saved all three members, and won the game.	printWon()	While loop in main stop iterating, and message telling users they have won is printed out.	Exited while loop successfully when getToBeSaved equals 0. System displayed: *Congratulations* *You saved all your friends!*
35	Free all memory allocated for objects in each space	deleteObject()	Successfully freed all memory allocated for each objects in a given space.	System displayed: *Deleting objects in Office.* *Deleting objects in Office.* *Deleting objects in Office.* This was the correct behavior as there were originally three objects in Office.
36	Free all memory allocated for each space	freeMemory()	Successfully freed all memory for all spaces. No memory leaked.	System displayed: *Deleted Office* *Deleted Lobby* *...* Valgrind reported that all memory were freed at the end of the program.