Name: Christopher Matian
OID: 933308644
Date: 12/1/2017

General Game Theme

So the game design is rather simple and nothing fancy. I didn't feel up to par going into this project with something massively complex. I decided on a riddle-based game that gave small hints to the player based on the room they were in. Several rooms would use the riddles to allow the player to enter a new room, or the player's would receive a key needed to unlock a chest in another room. Again, super simple and conceptually mundane.

The player can choose a name for the character and there's a "neat" little prize at the end of the game. Additionally, rather than a time-based limitation on the player, the player has a rather limited amount of HP (5 health). Each time they fail a riddle they lose a point of health, so it's somewhat important that they managed their health. There's 4 riddles total so you can avoid to get one wrong answer - afterwards you need to be really careful. The riddles themselves are pretty simple. If you've seen ***The Hobbit: An Unexpected Journey*** you'll probably recognize 3 of the 4 riddles immediately. There are paintings in each room used as clues to indicate what one of the answers might be. You can also use the cheat sheet that's printed in the main menu (before the game starts) to get through the game faster and verify that it works. Up to you.

Initial Design Strategy

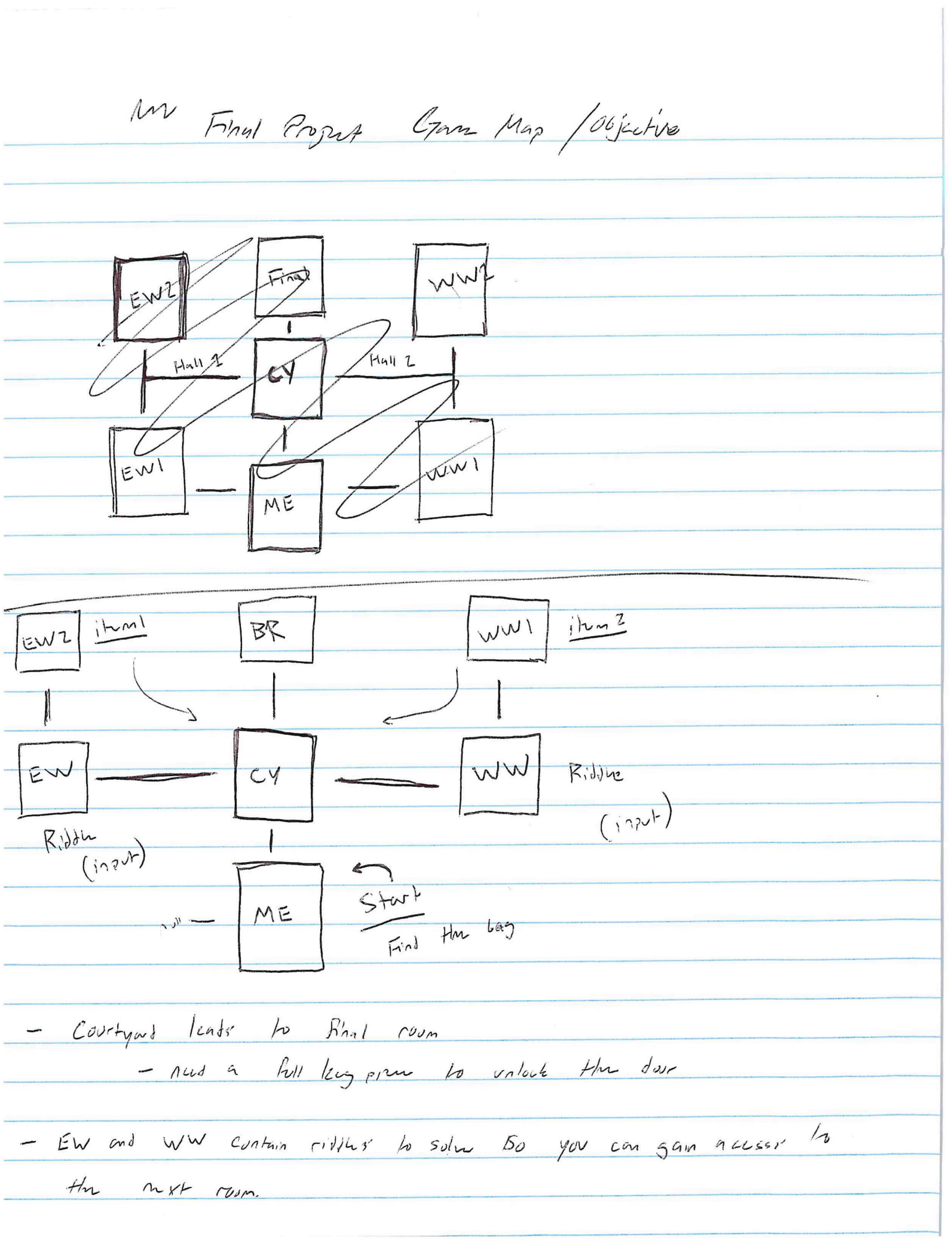
So beginning with this final project, the toughest thing I found was designing the overall structure of the program and implementing the RoomNodes/Space pointer structure. I stitched the project together room by room and ran various types of tests to check for whether or not the rooms were linked properly, and whether the rooms virtual functions were being called correctly.

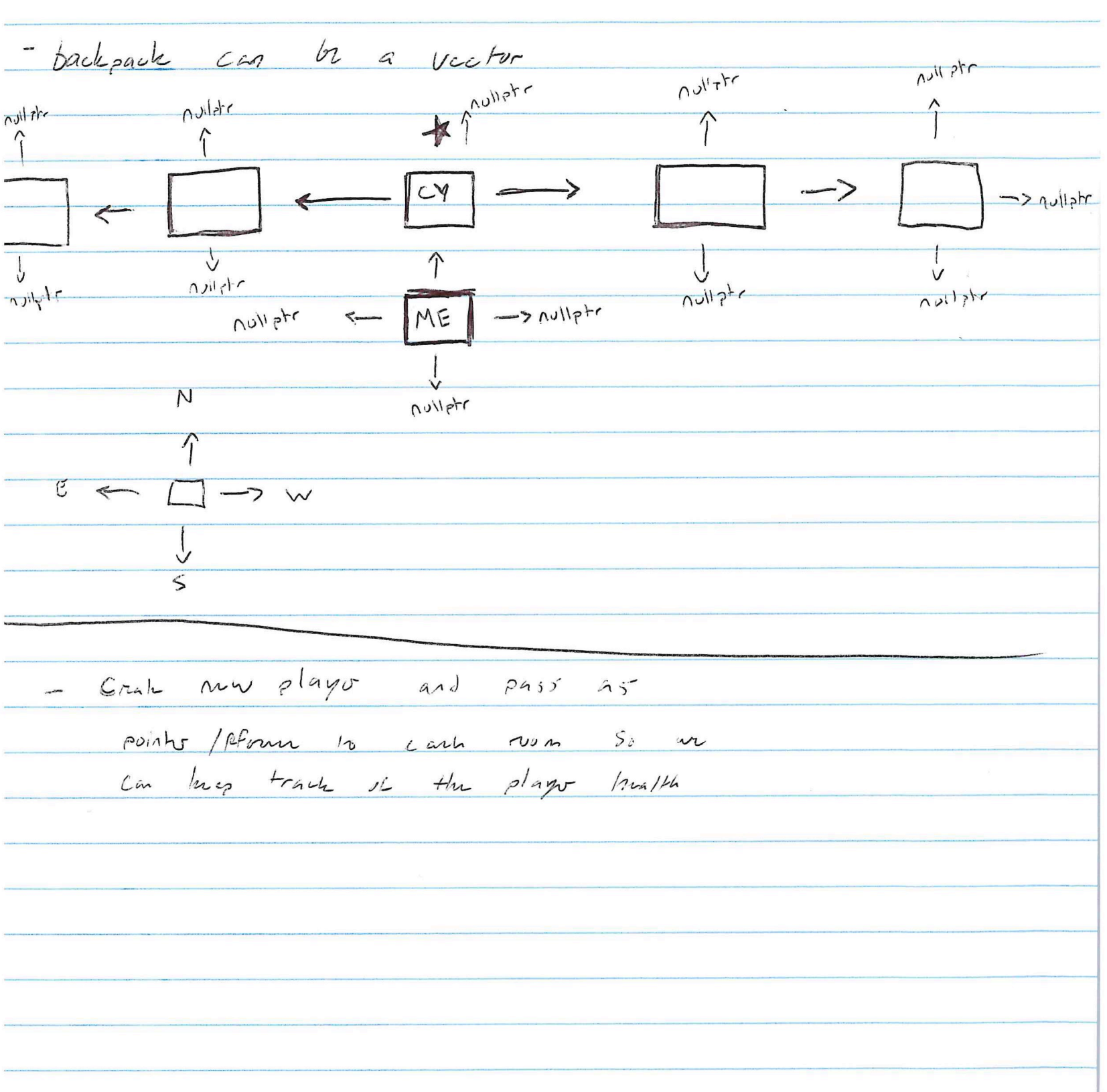
The general structure of the program would have the Space class function as an abstract class, and it would contain various pure virtual functions that would be leveraged by the derived room classes. In the physical code itself, 2 rooms don't use two of the virtual functions but have a spot defined for them - this was mostly because of a compilation error where I needed the function to exist even though it didn't do anything. I think there's another way of doing this but I'm not 100% sure of how to go about it, and I didn't want to fret over detail like that.

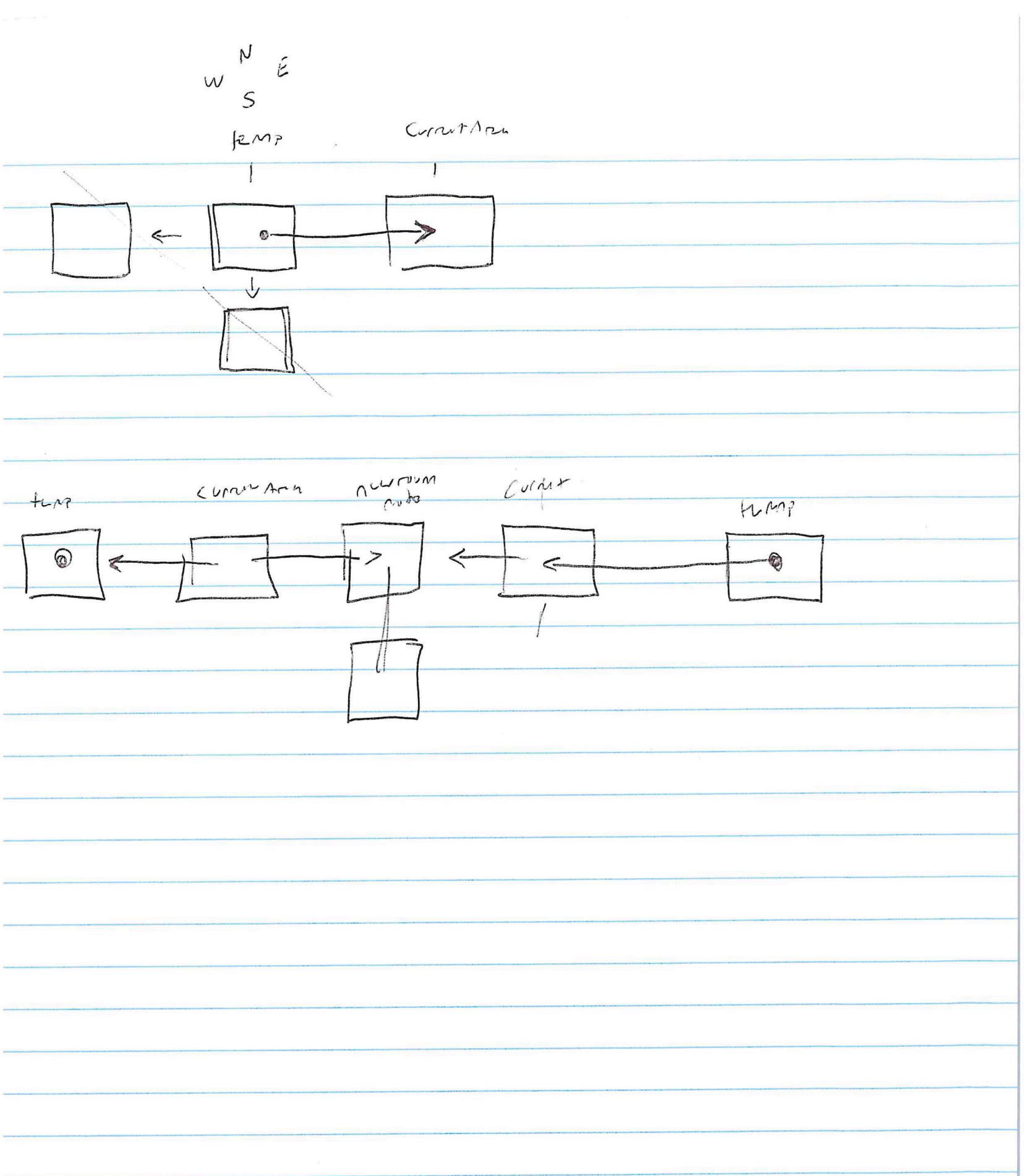
The main.cpp file essentially contains a basic menu where the player can start the game, view instructions, check the answers to the riddles, or exit the program. It's basically a simple do-while loop. The Game object is created when the player starts the game, and the Game enterNewRoom() function starts up which takes information from the object and starts the player off in a lobby.

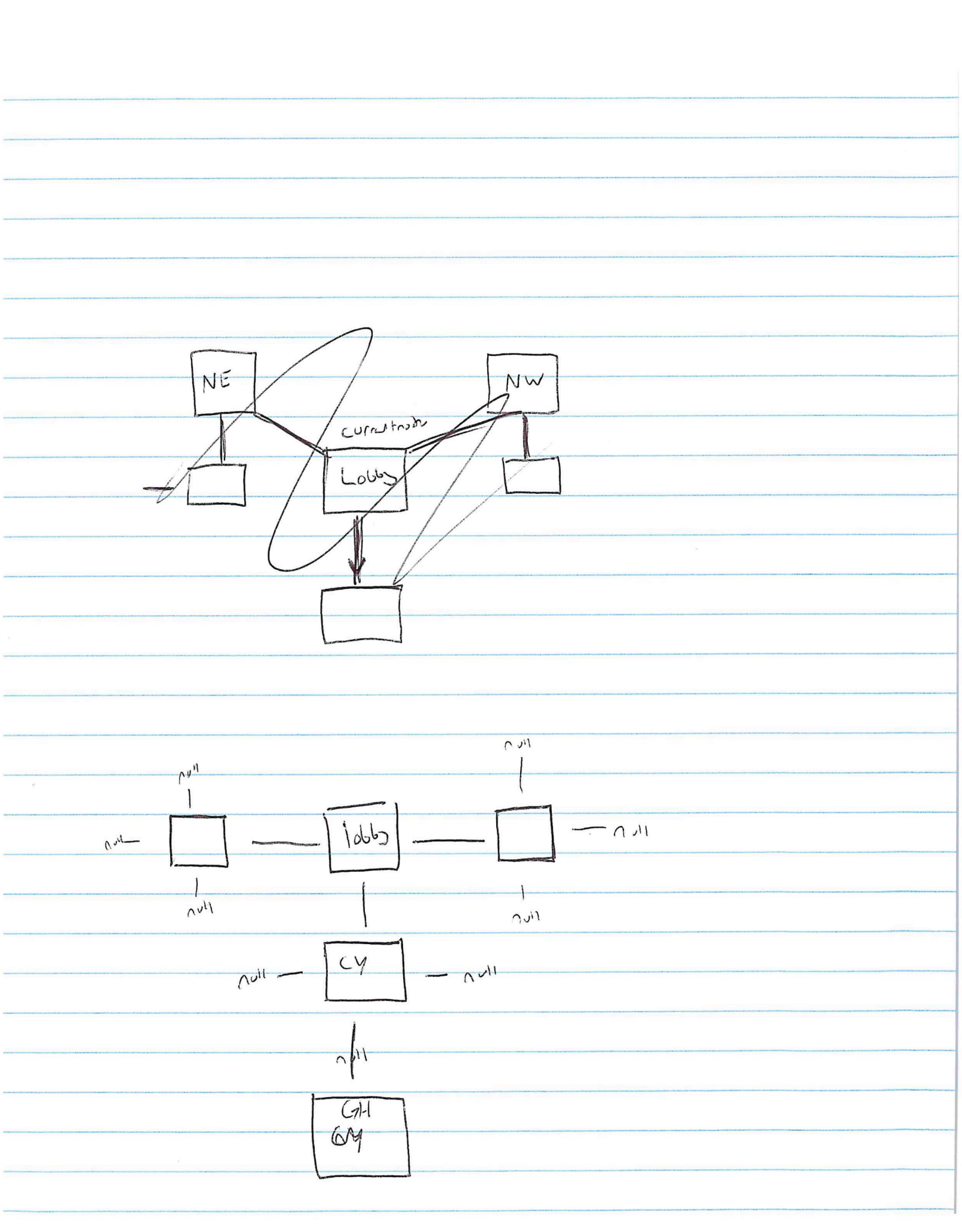
Overall, my design strategy revolved around using switch cases to control what the user was doing in each room. Additionally, to handle memory management of the rooms, I made sure that only the adjacent rooms were kept in memory - rooms that were beyond 1 room were deleted. This helps with resolving the headache of overwriting memory and losing scope of the address when trying to delete it much later. I didn't really like how I managed memory in project 4 because it was dirty and primitive. It was an absolute headache getting the game to repeat itself because the creatures would overwrite eachother and mess up the memory address stored in the vectors I had produced to solve the issue to begin with.

The following images are where I mapped out general thoughts and ideas of how I wanted the game to function:









The game layout changed a few times during development and I ultimately settled on using the following code block as the map of the zone:

```
[==================================== [ ZONE MAP ] ===================================]
[                                                                                    ]
[                                                                                    ]
[                                                                                    ]
[    W. Study        W. Hall             Courtyard          E. Hall          E. Study   ]
[   [=======]     [=============]     [=============]     [=============]     [=======]   ]
[   [       -     -             -     -             -     -             -     -       ]   ]
[   [       | *** |             ] *** |             | *** |             | *** |       ]   ]
[   [       -     -             -     -             -     -             -     -       ]   ]
[   [=======]     [=============]     [=====[-]=====]     [=============]     [=======]   ]
[                                          ***                                        ]
[                                          ***                                        ]
[                                       [===[-]===]                                   ]
[                                       [         ]                                   ]
[                                       [   YOU   ]  Lobby                            ]
[                                       [         ]                                   ]
[                                       [===[-]===]                         N         ]
[                                     Front Entrance                      W   E       ]
[                                                                           S         ]
[                                                                                    ]
[====================================================================================]
```

This same code block is used as a virtual function for the derived classes to copy and indicate where the player is in the game.

Issues and Complications / Design Changes

Surprisingly not as many issues as compared to project 4. I found this project a little easier to manage because we were at least given the ability to control what we wanted to do (within limits). I think there were two hitches I ran into that I managed to resolve.

First one was how to manage the player's health. This ended up being not too difficult as I had originally thought. I was initially going to make the player passed as pointer into each virtual function so that I could manipulate his health in there. Rather than do that, I created a reduceLife() function in the player class and used that to wound the player each time they messed up a riddle. Since the riddle returns a boolean T/F based on whether they answered correctly, I can just set the varying moveTo functions to handle when the player takes a hit. Since player is generated inside of Game, I essentially have immediate access to the player class.

The second and final issue that I had was my destructor for the Game object wasn't getting called corrected. I suspected that it was because I was using exit(0), and a quick search revealed what I suspected. exit(0) doesn't call destructors on exit. So, I had to alter the code to utilize a boolean flag + a while loop that let the game know when it was time to exit. The only time the game exits is when the player dies or when they unlock the chest for their prize.

Oh, another small issue I had was I forgot to add two delete statements for the Western and Eastern Hallways when they moved to their respective study rooms. This was causing the courtyard to remain in memory even though we had disconnected the pointers and set them to something else. So, when they were reconnected, it was with a new courtyard pointer and the older one was basically lost in the heap. No bueno.

Various Test Cases

Test validateBounds function for bad inputs such as negative values and letters. (Test 1)	-10	validateBounds(intent);	Should catch the bad submission and query the user again.	**`Please enter a value that corresponds with the menu item above.`**
Move from Lobby to Courtyard	N/A	moveToCourtyard();	Player should end up in the courtyard room	Player is in the courtyard room and the menu functions are correctly referencing the derived class
Move from Courtyard to Lobby	N/A	moveToLobby()	Player should end up in the lobby	Player is in the lobby room and the menu functions are correctly referencing the derived class
Move from Courtyard to East Hall	N/A	moveToEastHall()	Player should end up in the east hall	Player is in the East Hall
Move from East Hall to East Study	N/A	moveToEastStudy()	Player should be in the east study	Player is in the east study
Move from East Study to West Hall	N/A	Several move functions	Player should be in the west hall	Player is in the west hall and the nodes are properly managed as the player moves across several rooms.
Move from West Hall to West Study	N/A	movetoWestStudy()	Player should be in the west study	Player is in the west study
Player interacts with the chest in the courtyard without the first key	N/A	moveToCourtyard() recursion menu	Player should be told to go find the key	Player is unable to open the first lock and has to go find the other key
Player interacts with the chest and tries to open it but only has the first key	N/A	moveToCourtyard() recursion menu	Player is told to go find the second key even though the first lock is unlocked	Player opens the first lock but is told to go find the second key because the inner lock can't be opened yet.
Player exits the game through the Lobby (doesn't finish the game)	N/A	moveToLobby()	Player exits the game and the destructors are called which delete the created objects	Destructor is called and deletes several objects but fails to completely delete all of the objects. Exit(0) needs to be replaced.
Player finishes the game after getting both keys and unlocking the chest	N/A	moveToCourtyard() menu	Player retrieves their prize and the game closes.	Player earns the prize but exit(0) fails to call the destructors properly resulting in a memory leak. Resolved by getting rid of exit(0) and using something simpler.