Brian Huang
CS162

Project Reflection

This project overall was very tedious, I was able to get most of the functions working. I was not able to get the swans to move or the apple functions to work. Another small bug in my code is that when picking up excess keys or apples the item appears behind you instead of staying in its same position. Another small potential bug is the reading in of the file. The second dimension needs to be one larger than what it actually is. I am unsure if this is a bug or if there is a hidden new line character I am suppose to include. If it is not one larger my program still works but the player moves diagonally when trying to move up and down and the last line gets cut off.

Actor Class:
This class has a virtual function called action() which passes in the board. Since I could not get the swans movement working the only the player has a definition. This class also contains 6 int values. These values are for the starting position of the player and then the next and previous positions of the player.

Player Class:
This is where my program does most of the work. I used an overloaded constructor to initialize all the necessary values for the player to start out. The action() function for the player reads in an input which then does a specific action. For movement it store the previous position, checks if the next position is good the moves the Player using move(). For my input validation the program will work as long as a correct key is pressed. It will not allow "k" but will allow something like "dddddc." I made it this way so multiple moves can be taken with one input. For example inputting "wwwd" would move the player up three spaces and to the right one space. If the player does this mistypes are not punished and are just simply ignored. My moveCheck() function checks if the next tile the player wants to move to is a good spot or not. This function will increment the amount of keys and apples the player has and is also responsible for moving the player to the next level. Inside this function two simple function called addKeys() and addAppples() depending on the tile these will simply add keys or apples.

My move() function is responsible for the dropping of excess apples and keys. Because of the way I designed it I was not able to get the key or apple to appear in it original spot if the player already has the maximum amount. Instead I was able to get it to drop where the player was previously. This function is also responsible for sending the player back if the adjacent square is a swan. The sendBack() function simply move the player based on the input, and if the squares above, below or to the sides of the new spot is a S it will send the player back to the start.

Swan Class:
Could not get this class to work properly. It was mostly because I was getting errors when trying make an array of them. The swan also uses an overloaded constructor. It reads in the dimensions of the board then randomly generates numbers between the bounds of the 1 and the dimension. If the spot generated is a space it replaces it with a S.

Floor Class:
To read in the board I create the proper name using a stringstream. It first reads in the amount of floors and tries to open the biggest floor, if it is not there then the program exits. To read in the file I create two string that read in the first line. The reading for the first dimension stops when it encounters a space. When a space is detected it moves on to the next dimension. When this is finished it creates an array for the board. It reads each character doing specific things depending on the character read. For example if a S is read it will turn it into a space. (really not sure why this is required. If the swans spawn randomly what it the point in marking them in the map?) The print function just go through each index of the array and prints it. The Clean() function runs through the array and deletes everything necessary.

Game Class:
This class just stores all the information. The main function here is the play() function. It reads in a number for the amount of floors the game has. It then passes that number to the floor class which will initialize and print the board. The while loop has the player take his action then increments the counter. A swan will spawn every time the counter is a multiple of 30. When the level ends it will run cleaning functions to delete everything.

Main:
The main will output a message if the program is not run with proper command line arguments. If it is run correctly it will get the number of floors, create a game class and run the play() function in the game class. It will do this for the amount of floors specified. When a floor is finished it will decrement the number and pass it back into the play function to go to the next level.

Text Files:
I mainly used one textfile to text my program. I would just add to the file every time a new feature needed testing. There are two files to test for proper level changes.