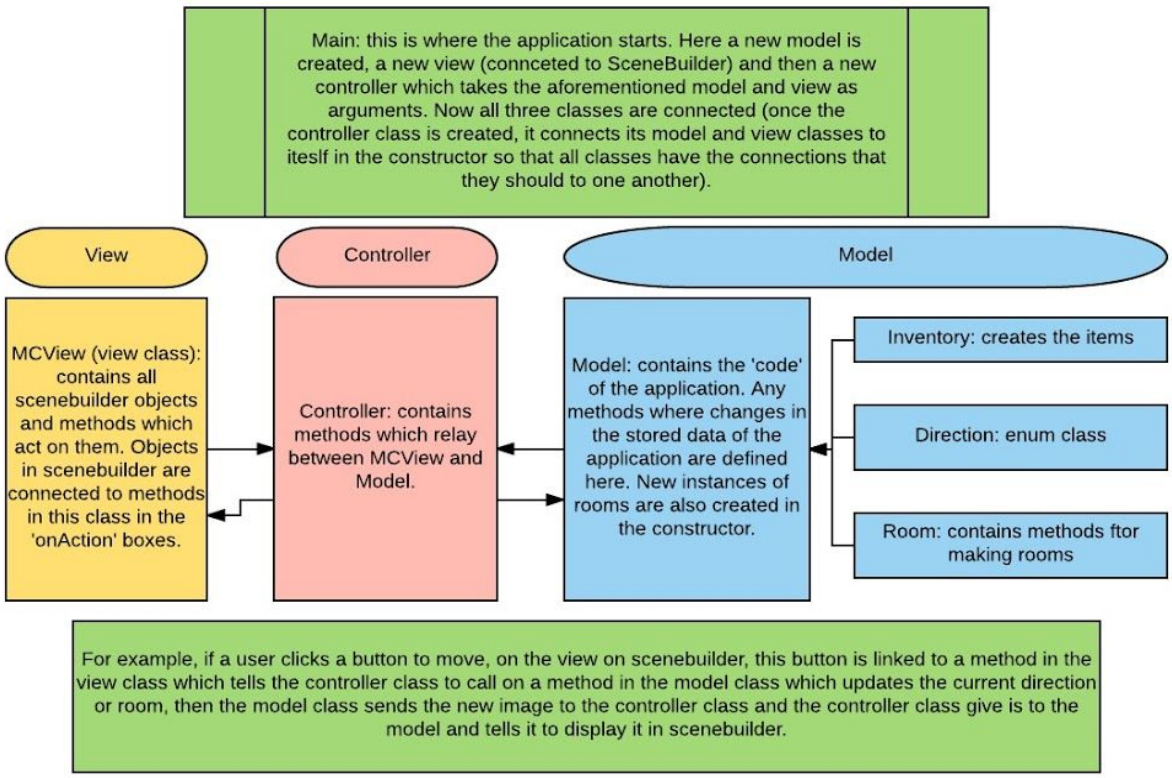
Worksheet

s1789372 Alexia McGregor

Question 1

Question 1: briefly describe your class design, including the function and relationships between the important classes. You may do this using text and/or diagrams, but the result must be clear and concise



Question 2

Briefly describe one or two significant choices that you had to make in the model design and explain why you chose your design over some plausible alternative

Originally I began with different classes, including the Room class, direction enum class, world (which was similar to my model class above except it talked directly to the 'view' which was the next class ->) and a 'worldcontroller' class (which had the

scenebuilder items so was sort of a 'view', but it also called methods from 'world', so it didn't really have a clear purpose as a class). This set up was functional up to a point. When I was attempting to add my items in, I was being forced to make static variables. For making new rooms, I had made rooms in the 'World' class and then called a new World() in the worldcontroller, but I could not figure out a way to do this for the items and kept on finding myself in different muddles. This is when I thought of modifying my existing design to a model-view-controller design so that I would not need to set things to static since the view (effectively 'worldcontroller' in my previous model) would not have to call things from a model class (which was 'items' and 'world' in my previous model).

The summary of all that is that instead of attempting (slightly unsuccessfully) to create classes which each dealt with their own aspect of the application which would have been functional but messy as the classes were interacting with lots of other different classes for different functions, I decided to implement an MVC design so that my code was cleaner, I did not have to create any static variables, and also so that if I hypothesize that I was working on this project in a professional environment, it would be easy to either keep the SceneBuilder interface and change the functionality of the buttons, and it would also be easy to keep the functionality of the buttons but change the view since these two classes are separate now.

Question 3

Add any extra comments that you would like to make about your solution, or the assignment in general. In particular, you should note any external resources from which you took code, and any significant collaborations with others.

I decided to modify my design to an MVC model after I had already written a significant amount of code. If I had thought of this from the beginning, I think I would have implemented it slightly different. For example, I set my images for my rooms and items in my model classes. I think it would be better practise to set these as some form of IDs and then store the actual images somewhere in the view class.

Some notes on my code/application:

My application lets you walk around rooms left, right and forward into a new room. You can navigating from the 'start' wall to the 'finish' wall and you can place down an item in a room by selecting the item in the drop down menu then selecting 'place hanging in room' and it will still be there when you return. You can also remove/pick up the images from a room. I made my application to have any amount of items you want to choose from from a drop down menu

comboBox, and I only wanted each room to allow you to place one item as a way of referencing the rooms.

If someone wanted to set a new room, they would need three lines of code: instantiate a new room, use the method to set their images and use the method to set its exits, all in the constructor of the model class.

If someone wanted to set a new item, they would need two lines of code: go to the inventory class (part of the model classes), and in the constructor, put a new name and image in the items hashmap, and then add the name to the inventory (so that it shows and is functional as an option in the combobox drop down menu).

In my application, it makes sense to have just one imageview. The inventory can hold an arbitrary number of items which means that the application can support an arbitrary number of items and you can display as many options for items as you want in the drop down menu of the combobox. I have set in my code that each room should only hold one item by using a queue (see setRoomObject() in class Room) because I only want one image associated with each room at a time (evident in application - each item relates to one room and you can set it down as you walk through each).

Referencing images used:

All the images used in my application came from advanced google search images with the usage rights set to 'free to use, share or modify, even commercially'. I accessed them all and set them in my application on 23/11/17.

The images were found on the google search were taken from from pixabay.com, publicdomainpictures.net, commons.wikimedia.org, staticflickr.com and freegreatpicture.com/.