Updates

In this document we will go through each class that ended up making it into our final program. We will describe how each of these classes changed from our initial program specification and prototype to the final version of the project.

General Updates:

Before diving into specific updates we encountered within each of the classes, we wanted to provide a general overview of our updates as a whole. Bluntly, we spent a ton of time setting up our program before even writing a line of code. We thought through all the classes that we would need, drew diagrams to help depict our thought process, and meticulously planned how the classes would interact with each other. As a result, we didn't have any huge updates in terms of overall structure to our program. We added a few classes, including Hint, myDS, and Puzzle, and we will discuss how those classes helped us to achieve our goal below. Other than those classes, the bulk of our updates in our classes included in the initial program specification were fairly semantic and more so in the details. We will discuss this below, but for further information on the classes, methods, and variables please see the attached program specification.

Main Class:

The Main class did not change all that much from what we had anticipated in our prototype. However, implementing the things that we anticipated the Main class doing took quite some time. We had to figure out how to use MouseListener and KeyListener effectively as well as navigate the difficulties of combining all of our different classes. In essence, our initial, very broad idea for Main was correct. However, at the time, we had no idea what form this would take or how challenging it would be to do so. It proved to be quite challenging, and the little things that we anticipated being rather simple became quite complicated with so much going on in our program and this class specifically. One example of such was that we initially planned to keep track of most of what was going on in our room with booleans, but this proved to be quite challenging. We instead decided to make specific objects to which we could point, such as currentArea, which allow us to navigate through different classes quite nicely. Additionally, we did not really know how we were going to check if user's inputs were sufficient to solve puzzles at first, but we ended up solving this issue by using a data structure and calling methods within that data structure. Clever solutions to problems like this helped to make Main more effective and easier to navigate in the end.

Room Class:

In a similar way to Main, our main ideas for how the Room class would function were correct from the start. This planning certainly helped us navigate some of the challenges that we faced down the road. Some of the details, however, of how we thought the room class would function

were a bit off. To begin with, we thought that we would have an arraylist of areas rather than an array (or 2 arrays) of areas. We decided to go with an array rather than an arraylist because our Area class was kind of like a linked list, which gave us capabilities that we wouldn't have had otherwise. Because of this, we could fairly easily insert areas into this list by adding an area to our array and we could add this area anywhere along our array. Additionally, the array made accessing specific areas a bit easier because calling areas[i] is slightly more convenient than calling areas.get(i), which may sound a bit ridiculous, but we thought would be helpful for writing such a large quantity of code. Finally, initially we thought that we would have an update room method which would go through and update the areas, but it turned out that we took care of this functionality in the Main class. Taking care of this in the main class was easier due to the fact that the Main class implements KeyListener and MouseListener, which allows us to take in user inputs and adjust the room accordingly. Additionally, unlike Pong, which we were basing the updateRoom method upon, our program does not have physics, so we were not changing the location of objects on the screen which would need to be done in some sort of updateRoom method.

Area:

From a far it may not appear as though our Area class changed all that much since the prototype or the initial project specification. This is due to the fact as a whole our idea of the code was fairly on point from the start. We have made some fairly drastic changes to Area, however. This included adding certain variables (isSolved) and methods (puzzleSolved, getSolved, setBottomRect()) as well as, perhaps the most drastic change, adding a new constructor. These methods and variables actually worked in tandem with this new constructor. The second constructor was useful in adding a second image to an area so that the image displayed in the area changed when the puzzle in that area was solved. Taking in this second filename in our constructor allowed us to change the image easily and not have to worry about changing every instance of area (as not all areas include a solved puzzle). These additional methods allowed us to incorporate puzzles being solved fairly easily. A more detailed description of these methods can be found in the attached program specification, which details each member variable and method fairly extensively. In essence, we added another level to our Area class by creating a different constructor, and therefore, adding to the capabilities of the Area class.

Puzzle:

Creating the puzzle class, a subclass of the area class, was very helpful from an organizational perspective. It allowed us to create objects which contained everything that an area would contain, as well as a boolean that tracked whether a puzzle was solved or not, and a corresponding solved image. This was convenient/important because instead of creating a bunch of areas for our puzzles and mixing them in with areas in the room, we could keep track of areas and puzzles separately. Additionally, the Puzzle class contained methods that overrode the

methods of the parent class, namely removing all of the hitboxes if a puzzle was solved, which we did not want to do when an area was solved.

Hint:
After having a couple of our friends try our escape room, we realized that the room might be a bit difficult for some people to complete--thus, the Hint class was born. The hint class allows us to print Hints for the user, depending on what area/puzzle they are in. We also implemented a feature where each area/puzzle will (when applicable) contain multiple hints that are randomly given to the user. This was done in an effort to help a truly stuck user make it through the room.

Rectangle:
Originally we thought we were going to implement the clicking and dragging functions within the rectangle class, but we later realized that those functions were better suited for the main class. We also never ended up needing to draw rectangles, so we were able to get rid of that method.

Runner:
There were no updates for the runner class, but having a runner that constantly listened to mouse inputs was a vital part of our program. Is was the backbone for all mouse interactions, allowing the user to move from area to area, and to solve puzzles.

MyDS:
A major update that we didn't originally anticipate was the use of the MyDS data structure. This data structure ultimately allowed us to check whether or not the user solved the puzzles successfully, in the correct specific sequence. In the end, we thought this was a pretty clever, clean, and succinct solution to the problem of verifying whether a user solved puzzles.

Inventory:
We were pretty much spot on with our original inkling as to what the inventory class would contain. As we said, the class contains a set of images, and a few boolean variables that we track. If a user has reached as spot in the escape room where they should receive an inventory item, booleans within the inventory class flip, and the item is printed in their inventory. This class was a concise way of implementing a feature we wanted in our program from the offset--a printable inventory for the user.

Pair:
No updates to note here. The pair class was essential for doing just what it sounds like it would do--keeping track of two ints. It was particularly useful for keeping track of coordinates.