Peyton and Nico

ESCAPE ROOM PROGRAM SPECIFICATION

(This updated version of the previous Program Specification gives a better description and more accurate description of the methods that we used in our code to help provide understanding of what is going on. It also provides some sources that we used along the way and some images of our code working.)

High Level Description

When we submit our final version of this project we hope to turn in a virtual version of an escape room. The escape room will feature a series of puzzles, riddles, and games that must be completed (often times in a specific order) to complete the escape room. To complete said tasks, the user must use their mouse and keyboard to interact with the escape room. The program will take in these user inputs, altering objects within the program accordingly, and changing which tasks are available at what time. (For example, new areas will open when certain tasks are completed). Further, we will be using still shots (shot with a high quality camera) as our images for the room, and point-to-move controls from the mouse for interacting with the room. The overarching purpose of this project is to offer users a fun, unique, and hopefully brain-teasing experience.

Detailed Description

Main Class:

1. The largest of our classes, the main class will allow us to move through our different areas and puzzles integrating them into one class. In a similar fashion to our Pong program, we will set the size of the screen and have the major components necessary to paint the screen in this class. We will want to implement KeyListener and MouseListener in this class.
2. In a similar fashion to our Pong project, we will want to have a static variable for the width of the screen, height of the screen and FPS of the screen. Although FPS may not necessarily appear important at first, we will want to use it when updating the screen when a user is moving an object across the screen. We will declare an instance of room in this class, and thus have the static variable r. This variable is static in order to use it in

static methods. We have boolean variables that are public to help us know where the user has clicked on the screen. Namely, if they've clicked to move right, to move left, to access a hitbox, to access a puzzle, or to go back to an area from a puzzle.

3. We will just have one constructor which helps set up our Room class. This constructor may tell the Room class where to put the inventory box, what images to use for the background, etc. It will also add the keyListener and the mouseListener (have not added much functionality to key listener yet but it will be important later).
4.
    a. public void keyPressed(KeyEvent e)
        i. The first three methods simply implement the interface of Keylistener, they return void and take in a key event.
    b. public void keyReleased(KeyEvent e)
    c. public void keyTyped(KeyEvent e)
    d. public void mouseClicked(MouseEvent e)
        i. This method basically waits until the mouse is clicked. Once the mouse is clicked we check if the point on which the mouse was clicked is within a hitbox and manipulate the room/area from there. It returns void but is a helper method for most of the methods in this class.
    e. public void mouseEntered(MouseEvent e)
        i. This method and the following three methods are necessary methods for using the interface MouseListener, but do not do much yet, they will be important later, however.
    f. public void mouseExited(MouseEvent e)
    g. public void mousePressed(MouseEvent e)
    h. public void mouseReleased(MouseEvent e)
    i. public void paintComponent(Graphics g)
        i. This method paints our room by painting the area that we are currently in. It is called in other methods if something is done by the user to manipulate the room in some way i.e. moving to another area etc. It takes in a graphics package and returns void, but it is very important at drawing the room/area/inventory.
    j. public boolean checkPair(Pair p, Room r)
        i. This checks a pair to see if it is within any of the hitboxes on the screen. It calls isIn in room in order to iterate over all of the hitboxes in the area and determine if the mouse clicked in any of them. It takes in a pair and a room and returns true if it was in any of them and false if it was not.
    k. public boolean checkLeft(Pair p, Room r)

        i. Once in one of the hitboxes, we must now determine which one the user clicked. This determines if the user clicked within the left hitbox and wants to move to the left room.
    l. public boolean checkRight(Pair p, Room r)
        i. Same thing for the right hit box.
    m. public boolean checkHitBox(Pair p, Room r)
        i. This method does the same but for the hitbox that is neither the right nor left hitbox i.e. one to go to a puzzle or grab something for the inventory.
    n. public boolean checkBottom(Pair p, Room r)
        i. This checks the bottom if the user is in a puzzle and wants to return back to the area he/she was previously in.

Room Class:

1. Similar to our world class in our Pong code, the role of our Room class will be to help facilitate the cohesion of our code. In this class, we plan to have an arraylist of Areas that we can iterate through and update, as well as methods to help draw the screen and methods to help update the screen.
2. As mentioned above, the room class will have an arraylist of Areas, which will allow us to move from Area to Area and store particular information about each. It would probably be advantageous to have an array list as opposed to an array because we may want to append areas to our area array if we want to expand our project. Likely here we will be sending in images, and neighbors for each of the areas in order to help make the user interaction with the areas better and in order to make it easier for us to move through the areas within the code. We may also want to have a variable for time or a stopwatch variable because our program is simulating an escape room. We can do so using the java.time library which allows us to access a stopwatch through duration (I believe). There also may be a better way to do this by using a delay, which is something that we will be looking into over the course of the next few days. We will want a height and a width int variable to keep track of the height of our world and the width of our world, likely for the purposes of not letting users drag something off of the screen. We will also want to have some font variable so that if we want to draw a String onto the screen within the Room class, we can using the font that we imported. It's going to be crucial that we track where exactly in the room the user currently is (ie, what area they're in). To do this, we'll use a simple array and an int counter variable, currentArea, to keep track of where they are. We will also want to declare some rectangles within our room class that we will assign as hitboxes for each area. Two of these rectangles we may want to keep constant for each area, a left and a right hitbox which will allow a user to go to the area to the right or the left area. The other hitboxes will be unique to each area depending upon where the puzzle or where the inventory is in the room. Perhaps, then we will want to only have the

right and left hitboxes as static hitboxes. Finally, we will probably want to have an arraylist for the inventory which will add images whenever the user finds something that can be stored in the inventory. This arraylist will be empty initially, but will contain images if the user clicks in certain hitboxes.

3. We will only have one instance and one constructor for our Room class because we will only have one room featured in our game. No new instances will be created by the user because we will have everything set before the user begins the game. This constructor will likely take in parameters for the width of the world the height of the world. Likely, what this constructor will do is set up the different areas and puzzles that we want to use. For example we may set up 3 areas, each with one puzzle and one with an inventory item. We will do so by adding three arrays to our arraylist and sending in information as well as declaring puzzles for each array. It may be easier for us to make some standard puzzles within our puzzle class which we can assign to the areas in the Room constructor.
4.
    a. public boolean isIn(Pair p)
        i. Very simple method. All this method does is take in a pair and iterate through the array list of rectangles in the current area to determine if the user clicked on a hitbox. If the user did click on a hitbox, it returns true, if he/she did not then it returns false. It is public so that it can be called from other classes. It is not recursive.
    b. public void drawRoom(Graphics g)
        i. This method draws our room by calling draw area on the current area that the user is in. It takes in a graphics package and returns void because it is only drawing something. It is not recursive and it is public so that it can be called from different classes.
    c. public void updateRoom(Graphics g)
        i. This method currently is empty, but could be useful in the future for implementing movement of object inside the room in a similar fashion to how the rectangles move in pong and how the balls move in keyboard spheres in pong. It would return void and likely have some kind of manipulation of a static pair for position and velocity. Perhaps since it is the room method, similar to our world method in Pong and Keyboard Spheres, it would call another update method within the Area class.

Area Class:

1. This class will represent the background workings of the images that are shown on the screen. Therefore there will be many instances that exist of the Area class (corresponding to how many areas in the room we end up having.) This number of instances will stay

constant, ie. no new instances (new areas) will be created after initially running the program. (Certain areas in the room will just be locked at the start of the game, and will be unlocked throughout. Although the number of instances will stay constant after initially running the program, the program will be expandable so if an individual (or we) want to add additional areas to the escape room at a later time we will be able to do so fairly seamlessly.

2.

isUnlocked: This variable will be of type boolean and will track if a certain area in the room can be accessed. This variable will be private--we don't want users to be able to access/update this variable without our consent. As the user completes tasks, this variable will be updated accordingly to track if it's "true" or "false" that the area should be unlocked.

Objects with which the user can interact that are in a given area: although obviously having far different names than this, these objects will likely vary between area to area and will likely actually be a combination of different types of variables. For example, we may have a chair next to a table which a user can move using the mouse. The chair, therefore, would need to have a number of variables that correspond to it. We would want the chair to have a position, chairPosition perhaps, which would be of type Pair and would store the x and y position of the chair. We would want chairPosition to be public so that it could potentially be accessed from outside the specific instance, but we could probably work with the chairPosition being private as well. We would also want a starting Rectangle which would be a hitbox for the square which would allow the user to move the chair by dragging it with a mouse. When the click releases, we would want the chair to move back to the initial position or into a final position. We would want a final Rectangle which would allow the user to place the chair in some other location and have it stay there. We would want these two rectangles to be private indicating that we did not want them to be changed throughout the program and that they were supposed to be permanent hitboxes. Finally, we may want a boolean chairInPlace which would be false if the chair was not in its final position and true if it was. Perhaps this boolean would be the key for the isUnlocked boolean that would unlock another area that could be accessed.

leftNeighbor & rightNeighbor: We will likely have neighbor variables which will be of the form area in order to move from one area to another. We will likely declare these variables in the constructor but set them after the variables have been created.

Puzzles: Each of our areas will have some sort of puzzles which will allow users to interact with the screen and these will be different for each area.

3. With our constructor, we would likely want to take in some background image and keep each background image associated with each specific area instance. Additionally, we will want to declare our neighbors for the area. For example, we want to keep track of the area to the left and to the right of any given area, so we can access an area's neighbor from within each instance.

4.

a. public Image loadArea(Graphics g, String filename)
    i. This method takes in a graphics package and a filename and loads an image of that filename and stores it to the area. This method is public so that it can be accessed from other classes. It is not recursive and returns an image that will be stored to the class.
b. public void drawArea(Graphics g)
    i. This method basically draws the area by drawing the image that we stored using the method above. It takes in a graphics package and returns void because it is simply drawing. It is public so it can be accessed from other classes, namely the room class, and it is not recursive.
c. public void paintComponent(Graphics g)
    i. This method paints the area, which was mostly useful for debugging purposes, to be honest we will likely take it out by the time the final project come around.
d. public void setRightNeighbor(Area neighbor)
    i. The setRightNeighbor method sets a right neighbor for our Area, turning our areas into a linked list. This is useful for going through the different areas. It takes in an area and sets it as the right neighbor for some other area. It is public so that it can be accessed from other classes and it is not recursive.
e. public void setLeftNeighbor(Area neighbor)
    i. Same thing as above for left neighbor.
f. public void setForwardNeighbor(Area neighbor)
    i. Same thing as above for Forward neighbor.
g. public void setBackwardNeighbor(Area neighbor)
    i. Same thing as above for Back neighbor.
h. public void createHitBox(int x, int y, int width, int height)
    i. This basically creates an extra rectangle and appends it to our linked list of rectangles. It takes in some x and y coordinate for a rectangle and the height and width of the rectangle and adds it to

the end of the linked list. Therefore, this method returns void and it alters the class variable that is our linked list of rectangles. It is not recursive and is public so that it can be accessed from other classes.

i. public void setPuzzle(Area inPuzzle)
    i. This method works in a similar fashion to our set neighbors method. It sets a puzzle for a specific area when called. Since a puzzle is simply an area class, we send in an area which is set to the area's puzzle. This method is public so that it can be accessed from other classes and it is not recursive.

Inventory Class:

1. While interacting with the objects within a given area or within a puzzle, users may find things that will be stored in their inventory, such as a key. These objects will be useful later when they interact with other puzzles that require the user to have certain inventory items to solve the puzzle. Interacting with a puzzle that requires an item in the inventory without the necessary item in a user's inventory will bring up a message that the user should look throughout the room to find an object that may be able to help them solve the task at hand.
2. For the inventory class we will want mostly boolean variables to determine whether we should print certain images to the screen which will be displayed below a line of text titled "Inventory." These booleans will also keep track of whether or not a user has the necessary inventory to complete puzzles that require certain items in the inventory. These booleans will be public so that they can be changed from the room class if a user clicks on a rectangle that adds something to his inventory.
3. There will only be one constructor for this class which will basically set all of the booleans to false. There will only be one instance of this class which will be made when the room class is initialized and the class will only change when the user finds an object for the inventory.
4.
    a. public void drawInventory(Graphics g)
        i. This method draws our inventory. It takes in a graphics package and draws a Rectangle with the word "Inventory" on top of it. If our user finds objects that he/she can store in the inventory, an icon will be drawn. This method is public so that it can be called from other classes. It returns void because it is simply drawing the inventory. It is not recursive.
    b. public Image loadInventory(Graphics g, String filename)
        i. This method takes in a graphics package and a string for the file's name from which we are loading an image. Basically, it loads in an image and

sets an image variable to that image. It is public so that it can be accessed from other classes and it is not recursive.

Pair Class:

1. This class will be very important for keeping track of individual pieces of information (position, velocity, etc.) that are integral to our larger classes (rectangle, for example).
2. x: This variable will be of type double. It will track the x-coordinate for a paired piece of information (position, velocity, etc.), and will be updated as the user uses the keyboard and the mouse.
   y: This variable will be of type double. It will track the y-coordinate for a paired piece of information (position, velocity, etc.), and will be updated as the user uses the keyboard and the mouse.
3. There will only be one constructor for this class--we only need to initialize the x and y components of the pair once, and we can update their values using the methods I will describe below.
4. Methods: These methods are very similar to the ones in keyboard spheres and that you can probably find in most pong implementations, basically, they allow us to work in R^2.
   a. public Pair add(Pair toAdd)
      i. Takes in a pair and adds it componentwise to our pair. Public so it can be accessed from other classes and not recursive. It returns the original pair after toAdd has been added to it.
   b. public Pair divide(int denom)
      i. Takes in an int and divides the pair componentwise by that int. Public again so it can be accessed from other classes and it is not recursive. It returns a pair after manipulation.
   c. public Pair times(int val)
      i. Very similar to the method above but does scalar multiplication on a pair. Public so it can be accessed from other classes and not recursive. Returns the pair after it has been multiplied component wise.
   d. public void flipX()
      i. This takes a pair and flips the x value of that pair (i.e. makes it negative). Returns the pair after the x value has been flipped. It is not recursive.
   e. public void flipY()
      i. Same method as method above but for Y component.

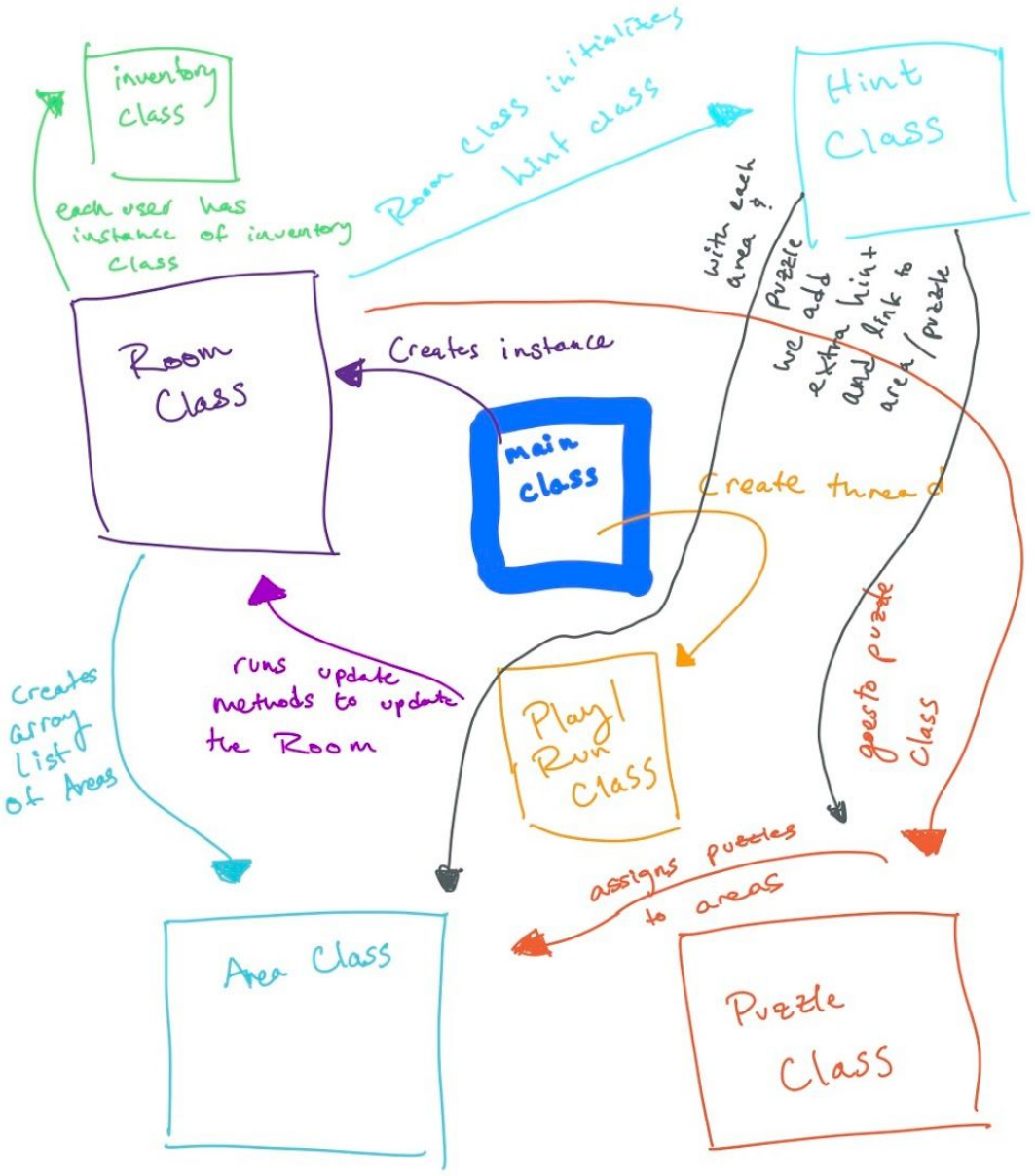
Rectangle Class:

1. This class will be used for keeping track of hitboxes within the areas. Specifically, hitting certain rectangles within an area will cause the user to move to a new area--this is how

we will handle moving around the room. In addition, this class combined with key listener will allow us to let the user interact with our puzzles, dragging and dropping, etc.

2. Member variables for this class will be similar to those used in our Pong project. We will track position and velocity of type Pair, height and width of type double, and possibly color of type Color if we decide we need to paint rectangles at some point. If other variables come up, we will add them in.
3. In a similar fashion to our Pong project, this class will have one constructor. The constructor will likely take in minX int, minY int, maxX int, and maxY int. These values will be ranges for which an individual can click and interact with the object on the screen. Thus, our constructor will take these values and save them with similar variable names for later use.
4.
    a. public void rectangleDraw(Graphics g)
        i. This method is mostly used for debugging purposes and for determining where hitboxes should go. It takes in the graphics package g and returns void because it is simply drawing a retangle.
    b. public boolean isIn(Pair point)
        i. The most important method in rectangle, this method takes in a Pair and determines whether or not it is in a given rectangle. This is particularly important for hitboxes. It is public so it can be accessed from other classes and it is not a recursive method.

Diagram (On Next Page):



(auxiliary classes... arrows TBD)

Pair Class

Rectangle Class


Inventory


Inventory

Sources:

https://pixabay.com/en/conference-room-table-office-768441/
https://way2java.com/multimedia/drawing-images/
https://docs.oracle.com/javase/7/docs/api/java/awt/Toolkit.html
https://docs.oracle.com/javase/tutorial/2d/images/loadimage.html
https://docs.oracle.com/javase/7/docs/api/javax/imageio/ImageIO.html
https://stackoverflow.com/questions/13038411/how-to-fit-image-size-to-jframe-size
https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html
https://developer.mozilla.org/en-US/docs/Web/API/MouseEvent/clientX
https://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseEvent.html
https://www.javatpoint.com/java-mouselistener
https://stackoverflow.com/questions/8755812/array-length-in-java
https://www.pexels.com/photo/kitchen-and-dining-area-1080721/
https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
https://www.pexels.com/photo/laptop-computer-884453/

https://www.pinterest.com/pin/423338433693867144/?lp=true