Peyton and Nico

ESCAPE ROOM PROGRAM SPECIFICATION

High Level Description

Our final version of this project is a virtual version of an escape room. The escape room features 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 takes in these user inputs, altering objects within the program accordingly, and changing which tasks are available at what time. (For example, new areas open when certain tasks are completed). Further, we used 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 allows us to move through our different areas and puzzles integrating them into one class. A large purpose of the main class is to implement KeyListener and MouseListener. Thus, in Main, we are able to take in user inputs and translate them to changes in the screen. Main is the workhorse of our program and brings all of the other classes together. One instance of the Main class is created when Main is run in the terminal. This instance of Main is created from within the main method in the Main class.
2. Member Variables:
    a. public static int WIDTH = 960;
        i. WIDTH is a variable that keeps track of the width of the screen. It is private so that it cannot be accessed or changed from other classes.
    b. public static int HEIGHT = 640; //height of screen
        i. Similar to WIDTH, HEIGHT keeps track of the height of the screen. It, too, is private so that it cannot be accessed or changed from the other classes.
    c. private Graphics g;

i. g is an instance of graphics that allows us to draw different objects to the screen. It is private, but it is not necessary that it is private. We did not see any major advantages, however, to making g public, and it could be advantageous that users could not access this instance of graphics from other classes.

d. private int FPS = 10;
   i. FPS keeps track of the frames per second at which objects are drawn to the screen. This variable was very important for listening to whether or not the mouse was in one of the hitboxes that displayed an arrow when dragged over. Due to the fact that we do not need super smooth graphics, we thought that an FPS of 10 was sufficient for updating the screen. The variable is private so that it cannot be accessed from other classes.

e. public static Room r;
   i. r is an instance of room that the user is in. Each time Main is run a new room is created which tracks a number of different variables. It is public so that it can be accessed from other classes, thought it is likely not necessary that it is public given our implementation.

f. public boolean inPuzzle;
   i. This boolean tracks whether or not a user is in a puzzle. It is important because we want to know how clicking on a specific hitbox will change the screen. Puzzles act slightly different than areas, so this is an important distinction to make. It is public so that it can be accessed from other classes, thought this is not necessary with our current implementation.

g. public boolean mouseInLeft;
   i. This boolean tracks whether or not a user hovers the mouse over the left hitbox. This boolean is important because if the user does this, we want to draw a left arrow to the screen. It is public so that it can be accessed from other classes.

h. public boolean mouseInRight;
   i. This boolean does the exact same thing as mouseInLeft, but for the right side of the screen and , therefore, the right arrow.

i. public boolean mouseInBottom;
   i. This boolean does something fairly similar to the previous two booleans, but rather than operating when the user is in an area, the mouseInBottom only applies to puzzles so that the user can move backwards from a puzzle. Again, it is public so that it can be accessed from other classes.

j. public static Main play;

i. This is the instance of Main that is created in the main method of Main. It is what we use throughout the program to run everything. It is public so that it can be accessed from other classes.

k. public Point p;
   i. p is a point that stores the location of the mouse temporarily before it is converted to a pair(the pair below). It is accessed in the static method listenToMouse(). Please note that in order to make sure that the mouse position was being tracked in relative terms, we had to subtract the position of the JFrame because a typical pointer position gives an absolute position, not one relative to the JFrame. It stores the mouse location and is important for checking whether or not the mouse is hovering over the right or left hitbox. It is public so that it can be accessed from other classes, though I do not believe we ever access it from other classes. We need to use a point rather than a pair because the method that we call (get location()) on the mouse returns a point not a pair. There do not seem to be apparent benefits of making it private.

l. public Pair mouseLocation = new Pair(0,0);
   i. Because of the way in which our other methods work, we need to convert the point to a pair in order to run the coordinate through checks. Thus, we used mouseLocation to take the values from the point p above and put them in pair form. The variable is public so that it can be accessed from other classes if necessary

m. public boolean correctPawn = false;
   i. correctPawn stores whether or not the user selected the correct pawn to drag. If the user did, the program listens and if the user drags the pawn to the correct space, the chess puzzle is solved. It is public so that it can be accessed from other classes. It is not currently accessed from other classes, but having this ability does not hurt.

n. public MyDS ds = new MyDS();
   i. This variable is an instance of a data structure which is helpful in determining whether or not the keyboard puzzle is solved and whether or not the riddle puzzle is solved. It is public so that it can be accessed from other classes, which is not currently done.

3. There is just one constructor for the Main class. This is because we want each Main instance to do the exact same thing, add mouse listener and keyboard listener and prep the screen for printing.
4. Methods: (None of these methods are recursive)
   a. public void keyPressed(KeyEvent e)

i. keyPressed takes in a key and determines what the user is trying to do. If the user presses an h, the user likely wants to bring up a hint or get rid of a hint. It also is necessary in facilitating two of our puzzles, namely the keyboard puzzle and the riddle puzzle. Users can type in their answers and keyPressed takes in these inputs and determines (with the help other methods) whether or not the user is putting in the correct inputs. It is public so that it can be accessed from other classes and it is non-static so that it is run on an instance of Main.

b. public void keyReleased(KeyEvent e)
   i. keyReleased is a necessary method in order to implement KeyListener, it does not do anything in our program.
c. public void keyTyped(KeyEvent e)
   i. Similarly, keyTyped does not do anything in our program, it is only necessary for Main to implement KeyListener.
d. public void mouseClicked(MouseEvent e)
   i. The following links helped us determine how to use mouse events and MouseListener in our program
      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
   ii. 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. It is public because there does not appear to be a compelling reason to make it private.
e. public void mouseEntered(MouseEvent e)
   i. mouseEntered is necessary for Main to implement MouseListener, but it does not do anything in our program.
f. public void mouseExited(MouseEvent e)
   i. mouseExited is also necessary for main to implement MouseListener, but does not do anything in our program.
g. public void mousePressed(MouseEvent e)
   i. mousePressed is mostly useful to determine if the user clicked on the correct space in the chess puzzle. If the user pressed on that space, we check if the user released the mouse in the correct space(below). This method is public and non-static so that it can call instance variables fairly easily and can be accessed from other classes.
h. public void mouseReleased(MouseEvent e)

        i. As mentioned above, mouseReleased is used to determine if the user released the mouse in the correct space in the chess puzzle. If the user does this, it tells that puzzle that it is solved and repaints the screen. It also puts a pick in the user's inventory. It is public so that it can be accessed from another classes and it is non-static so that it can be called on instances of main.
    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.
    o. public static void main(String[] args)
        i. The main method sets up the program to run. It creates an instance of main and does some of the background work to get the screen set up. It returns void and is static because that is necessary for the main method.

Room Class:

1. The Room class was created to help facilitate the cohesion of our code. The Room would be where all of the areas and puzzles lived, kind of like the real escape room. Inspiration for Room came from World in Pong which helped facilitate gameplay. This class helped to move through areas and keep everything in order. In addition, most of the brute force

was done in the Room class, which included loading and initializing all of the areas as well as setting some of the hitboxes (not right or left hitbox) for the areas. For this reason, the room class looks a bit messy, but it is necessary to our program that we load in these areas as well as well as initialize these hitboxes.

2. Member Variables
    a. Graphics g;
        i. This member variable is just an instance of graphics which is sent as an argument to create a number of different instances in the room class. The variable is not public or private as it does not need to be accessed from other classes, but this is not integral to the program, so we did not feel that it was necessary to make it private.
    b. int height;
        i. The int height contains the height of the room (or the height of the screen). This is sent in by the Main class when the room is constructed. Again, there do not appear to be major advantages to making this public or private, so it is neither.
    c. int width;
        i. This int contains the width of the room. Very similar to height above, but with width of screen rather than height of the screen.
    d. public Area currentArea;
        i. This area stores which area that we are currently in. It does so, so we know which area to draw to the screen when printing the room as well as where the hitboxes are in the room and what the currentArea's neighbors are. currentArea is integral to the functioning of Main and is public so that it can be accessed by outside classes.
    e. public Area areas[];
        i. This array of areas stores all of the main areas in our room. In other words, it stores all of the screens (that are not puzzles) that the user interacts with in the room. This array is helpful for a number of reasons. For one, it is useful for declaring all of the Areas and setting all of their neighbors. Additionally, it is helpful for accessing these areas from outside the class, including Main. Therefore, it is a public variable.
    f. Image leftArrow;
        i. This image stores the image that holds the left arrow for the screen. Thus, when we want to access this image in our drawLeftArrow() method, we can. It is not public or private because we do not need to access this image from other classes, but at the same time it is not necessary that it is not accessed, so there are not major advantages of either.
    g. Image rightArrow;

i. This image does the same as leftArrow but stores the image of the right arrow rather than the left arrow.

h. Image downArrow;
   i. Likewise for the down arrow that apprears on the screen when a user scrolls over the bottom of the screen in a puzzle.

i. int numAreas = 12;
   i. This number will be used to declare the size of our array, i.e. how many areas we will have in the array. This makes it easy to change the number of areas at any given time. It is not public or private because there do not appear to be major advantages of either. If we want to access the size of the areas array outside of the class we can call areas.length rather than grabbing this member variable.

j. public Area supArea[];
   i. This array contains areas that contain the puzzles (and one step between a puzzle and an area) that is in the room. It is named supArea for supplementary areas. This array is public so that it can be accessed from other classes.

k. public Area previousArea;
   i. previousArea, similar to currentArea stores the area that the user was previously in. This is helpful for when a user is in a puzzle and wants to move back to an area, the currentArea just changes to previousArea. It is public so that it can be accessed from outside classes.

3. We will have one instance and one constructor for our Room class because we 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 takes in arguments for the width of the world the height of the world. This constructor does the brute force work of setting up all of the areas and puzzles that are in our room. Additionally, the constructor adds all of the hitboxes to these areas. It also loads in the images for the left, right, and bottom arrows.
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 drawLeftArrow(Graphics g)
    i. This method takes in an instance of graphics and draws the leftArrow to the screen. From the Main method, this method is called if the user hovers the mouse over the left side of the screen indicating that the user can move to the left. It is public so that it can be called from other classes and it is nonstatic so that it can be called on a specific instance of room.
d. public void drawRightArrow(Graphics g)
    i. This method does precisely what drawLeftArrow does except for when the user hovers the mouse over the right side of the screen this method is used to draw the arrow to the right side of the screen.
e. public void drawBottomArrow(Graphics g)
    i. Same idea once again, if a user is in a puzzle and hovers the mouse over the bottom of the screen, this method is called to draw the downwards arrow to the screen to indicate that the user can move back to the previous area.

Area Class:

1. This class holds the individual screens with which a user interacts. Many instances of this class will be called for each individual area that is created within the room. These instances work together to create the escape room.
2. Member Variables
    a. public static final int BOX_WIDTH =(int)(1.2* 960);
        i. This variable stores the screen width for the area, i.e. how wide the images should be. It is public so that it can be adjusted from outside the class if necessary.
    b. public static final int BOX_HEIGHT = (int)(1.2*640);
        i. Same description as above, but rather than being the width of the area/screen it is the height of the screen/area.
    c. public Image image;
        i. This image stores an image associated with an area. Each area has at least one image that is printed to the screen when the user is in that area. Some images have more than one, as we will see below. This variable is public so that it can be accessed from outside of the class.
    d. public Image image2;
        i. This second image stores an additional image with an area if the image displayed with an area changes when the puzzle in that area is solved. This image is only created if another string with a file name is sent into the

constructor, indicating that this area has a second, solved image associated with it.

e. public static Graphics g;
    i. This is a public instance of graphics, mostly used for debugging, but also can be called in methods or sent as an argument to other methods that are in puzzle, inventory, hint, etc.
f. public int num; //number associated with each area
    i. This num is a number that is associated with each area, like an index. When each area is declared, we send in a num to store with that area so that we can have a way to identify which area is which from outside of the class. This variable is public so that it can be accessed from outside of the class. We use num a lot in main to determine which area the user is in because we may want to alter our behavior depending upon which area the user is in.
g. public Area leftNeighbor;
    i. This variable helps create a linked list of areas, which is useful in a number of ways. If a user wants to move to the left, left neighbor allows us to do so by simply saying that our current area is now our left area. Neighbors are integral to our program in connecting each of the areas within the room. This variable is public so that it can be accessed from outside of the class.
h. public Area rightNeighbor;
    i. Same thing as above, but this creates a two way linked list so that each area has two neighbors, which is helpful if we want to move to the left or to the right.
i. public Area puzzle;
    i. This public variable stores the puzzle associated with an area. If an area does not have a puzzle, we simply do not set the puzzle of an area, but this gives us an easy way to navigate through the areas and puzzles. It is public so that it can be accessed from other classes.
j. public Area myArea;
    i. Used more in testing, this associates a puzzle back to the area for which it is a puzzle. It is public so that it can be accessed in other classes.
k. public boolean isSolved;
    i. This boolean tracks whether or not an area has been solved, i.e. whether or not we should display the solved image for the area. This boolean is public so that it can be accessed and changed from outside classes.
l. public boolean hClicked = false;

i. This boolean is to track whether or not the key h was clicked, which determines whether or not we want to display a hint. This variable is changed from the main class if h is clicked and, thus, the boolean is public so that it can be accessed from outside classes.

m. public Hint myHint = new Hint(960,640,"no hint");
   i. This member stores the hint for a given area. If there is no hint for an area, we set the hint to be no hint. If there is a hint, however, we set a different hint and get rid of this original no hint Hint.

n. public static Inventory myInventory = new Inventory(BOX_WIDTH,BOX_HEIGHT, "pick.png", "lock.png");
   i. Declares a new inventory for the Area class. This is a static variable, so all of the areas have the same inventory, which is useful because being in a different area should not change an individual's area. This is public so that it can be accessed from outside classes, namely main.

o. public ArrayList<Rectangle> rect;
   i. This arraylist of rectangles is used to store the hitboxes for any given area. An area typically has 2 hitboxes, the right and the left hitbox, and then some other hitboxes which could be to access puzzles, or within a puzzle to attempt to solve the puzzle. We used an arraylist over an array so that we would not be constrained by the set size of an array. Additionally, we could add and remove hitboxes fairly easily with an arraylist which allowed us to custom tailor hitboxes to each area.

3. There are two constructors for the area class. One of these constructors is for areas that do not contain puzzles that change the image displayed by the area when solved and the other is for areas that contain puzzles that change the image displayed by the area when solved. The first constructor, therefore, takes in the arguments of a graphics package, a filename of the image, and an index which is mapped to a num. The second constructor takes in an additional argument of a second filename for the solved image and loads in that image, storing it in image 2. The first constructor is called from the second constructor, so other than this, they are identical. Basically, the constructor loads in an image and stores it in the image variable. It then creates an arraylist of size 2 to which it adds the left and the right hitbox. It then sets the num equal to the index sent in.

4. None of these methods are recursive and they all are non-static so that they can be called on individual instances of areas rather than on the area class as a whole. This is due to the fact that we want these methods to change something about an individual area not the area class.

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. We learned about loading images into a program and using images in general from the following links.
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

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 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.
g. public void puzzleSolved()

i. This method is called if the puzzle associated with an area has been solved. Basically what this method does is swag the image typically stored with an area to the solved image that is stored with an area, which changes the display. The method is public so that it can be called from other methods, specifically main.

h. public boolean getSolved()
   i. This method is called to get the value of isSolved. Although not necessary because isSolved is public, it is a logical command that we call occasionally to get the isSolved value. It is public so that it can be called from other classes.

i. public void setBottomRect()
   i. This method sets a bottom hitbox for a puzzle. We decided to create this method in order to get rid of the redundancy of calling the createHitBox with these values every time. The way in which our code is set up, it is necessary to add the bottom rect after all other hitboxes have been added which is why we do not put the bottom hitbox in the puzzle in the puzzle constructor. It is public so that it can be called from other classes. This method is called a number of times in Room.

Puzzle:

1. The puzzle class is a subclass of the area class. This is because, as is the case with areas, we wanted puzzles to have many of the same variables that are contained within an area (an image file, an index, etc.) and similar methods to the area class. The major distinction between the puzzle class and the area class is the way in which a puzzle is solved. If a puzzle is solved, puzzleSolved() is called which overrides puzzleSolved() in the area class. In the Puzzle class, puzzleSolved() removes all of the hitboxes from the puzzle other than the back hitbox which is done so that the user can no longer interact with the hitboxes, which would be a potential threat to the robustness of our code. Like the Area class, there are a number of instances of puzzle while our code is running. This is due to the fact that we have a number of puzzles for the user to solve/interact, each requiring its own instance of the puzzle class.
2. In addition to the variables included in main, we have two additional member variables in Puzzle.
   a. public boolean isSolved;
      i. This boolean tracks whether or not the puzzle has been solved. A major distinction between this isSolved and the isSolved in Area is that isSolved in the puzzle class is set to false in the constructor. It

is set to false in the constructor due to the fact that Main uses the condition !isSolved for some of the puzzles, which we want to guarantee is true if the puzzle has not yet been solved. Thus, we set it to false in the constructor. This boolean is public so that it can be accessed from outside classes.

b. public Image solvedImage;
    i. solvedImage is a variable to track the solved image file. Likely we do not actually need this variable, because we could store the second image within the area class, but it is reassuring to have it in the puzzle class.

3. There is only one constructor for the puzzle class which is very similar to the area constructor for the area that has a puzzle that changes the area image. All puzzles will be identical in macro structure, and thus, we only need one constructor.
4. Methods:
    a. (@Override) public void puzzleSolved()
        i. As mentioned above, puzzleSolved() overrides the puzzle solved method for the area class. Rather than just changing the image, this method also removes the hitboxes other than the bottom hitbox in order to prevent the program from being susceptible to user input error. This program is public so that it can be accessed from other classes. It is not recursive
    b. public boolean getSolved()
        i. This method does the same as the getSolved method in the Area class, but it returns the isSolved for the puzzle, which is set to false in the constructor. This method is public so that it can be accessed from outside classes.

MyDS:

1. This data structure's main function is to solve puzzles. In particular, it was born out of the following problem--we wanted a way for the puzzle to solve a puzzle if and only if they typed/clicked certain keys/areas on screen in a specific order. With the use of a data structure, we were able to do something similar to checkTrojan from lab7, checking to see if a puzzle was solved any time a user put in new input. There is one instance of the myDS class in our program.
2. This data structure functions by creating a linked set of nodes that contain salient information. Each node has a number (num) that's associated with it, and a linked node (next) that it points to. With these variables we are able to create sequences of nums that correspond to user input. We then iterate down the data structure, keeping track of the

sequence of nums to check whether or not the user has indeed solved the puzzle (clicked/typed in the correct sequence.)

3. The constructor for this class is simple. It creates an end node that contains null. All other nodes are built off of this end node.
4. A couple methods included in MyDS are superfluous for this project, but we decided to include them anyway, because we could see them being useful if we wanted to make enhancements to our escape room down the line. Such methods include peek(), pop(), toString() and length(). (One could, for example, imagine implementing a feature where users could see what their current input for the piano puzzle is. We decided this and similar enhancements weren't essential for the project to turn in, but could see ourselves making enhancements when we have more time.)
    a. public void append (int toAppend)
    This method serves to append new nodes to our data structure. It takes in an int (toAppend) and creates a new node with that int stored in num. It then points to the previously end node in the data structure, and stores that node in next.
    b. public boolean checkPiano()
    This method serves to check whether or not the user has solved the piano puzzle or not. It works by iterating down from the end of the data structure, creating a string of seven ints. It then checks to see if that string is equal to "1 2 3 4 5 6 7 ". If that is true, the piano puzzle has been solved successfully, and the method returns true.
    c. public boolean checkRiddle() Similar to the checkPiano method above, this method serves to check whether or not the user has solved the riddle puzzle or not. It works by iterating down from the end of the data structure, creating a string of six ints. It then checks to see if that string is equal to "1 2 3 4 5 6 ". If that is true, the riddle puzzle has been solved successfully, and the method returns true.

Inventory Class:

1. Our inventory class allows us to provide the user with a graphical depiction of the items they currently possess in their inventory. In our escape room we only have two items the users can possess, but with the use of this inventory object, we can track whether or not each of the two objects should be printed or not yet printed in the inventory. (One could imagine a more extensive version of this, tracking whether or not many more items should/shouldn't be included in the inventory).
2. The inventory class contains variables that fall into two categories: variables that handle graphics, and variables that track whether or not an item should be in the users inventory. Ints x, y, width, and height are all responsible for setting up the eventual printing of the inventory--they keep track of where on the screen and how big we want to print the 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 pieces of information that correspond to each other, for example, coordinates. These pair objects are integral to our larger classes (rectangle, for example).
2. x: This variable is of type int. It tracks the x-coordinate for a paired piece of information, and can be updated with methods we will describe below.
   y: This variable is of type int. It tracks the y-coordinate for a paired piece of information, and can be updated with methods we will describe below.

3. There is only 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 described 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.

Hint Class:

1. Our hint class allows us to guide the user along through our escape room. If, at any time, they decide to press h, they recieve a hint on screen. Each hint corresponds to an area/puzzle, and by keeping track of Hint objects, we are able to print out the correct hint for a corresponding puzzle, ultimately helping to guide stuck users through the room.
2. As is the case with the inventory class, the Hint class has variables that fall into two categories: variables that handle graphics, and variables that track where/how many hints belong in an area/puzzle. Width, height, chalkboard, and toPrint take care of the graphics, handling the way in which we actually print the hints on the screen. These variables set where we want to print on the screen, they set the font of the the string we want to print, and they set the string toPrint itself. The variables that take care of actually tracking where hints belong/how many hints there should be for a given area or puzzle are the array hints, the boolean moreHints, and the instance of random rand. These variables track an array of multiple hints for areas/puzzles (when applicable), a boolean that tells us whether or not a puzzle has more than one hint, and a random variable to get a random hint from the array of hints.
3. We have two constructors, one for areas/puzzles with one hint, and another for areas/puzzles with multiple hints.
4. There are two methods in this class. The differentHint() method sets the hint toPrint to a random new hint within the array, used for areas/puzzles with multiple hints. The drawHint() method draws the hint toPrint on the screen.

Rectangle Class:

1. This class is used for creating hitboxes within the areas, as well as telling whether or not a given user mouse click is inside or outside of a given rectangle.
2. Member variables for this class include an int location x and y, and an int width and height.

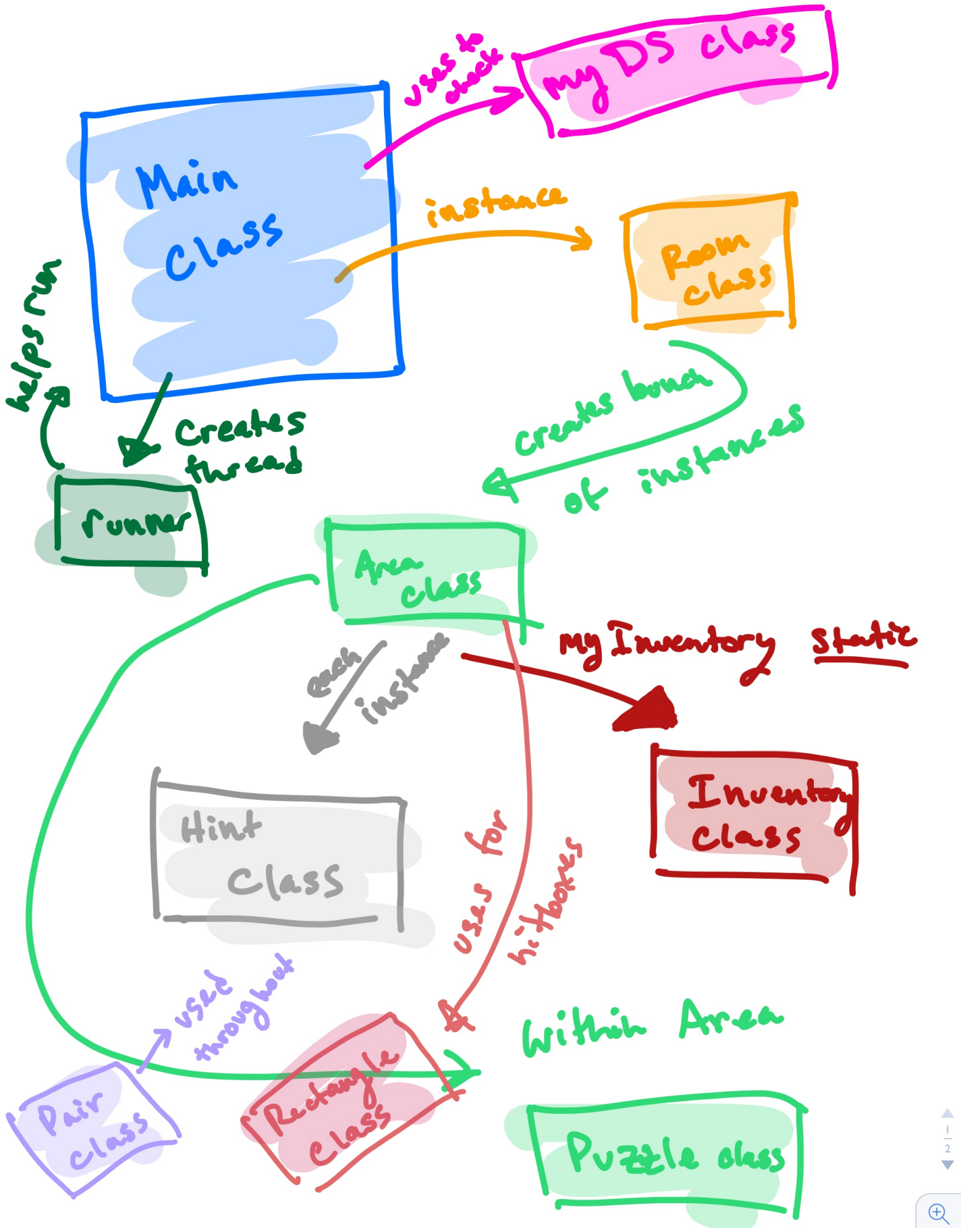
3. This class has one constructor. The constructor takes in two x and y int coordinates, and an int height and width.
4.
    a. public boolean isIn(Pair point)

    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.

Runner Class:

1. This is a runner class whose main function is to constantly listen to mouse inputs from the user and reprint the screen accordingly.
2. This class contains no variables.
3. There is no constructor for this class.
4. The important method for this class is the "run" method which is public simply contains a while loop that repeatedly listens for inputs from the mouse.

Diagram (On Next Page):


Main Class
uses to store
my DS class
instance
Room class
creates bunch of instances
helps run
Creates thread
runner
Area class
each instance
my Inventory
static
Inventory class
Hint Class
uses for hitboxes
used throughout
Within Area
Pair class
Rectangle class
Puzzle class

Sources:

Rotherwas Room(17th Century)
Mead Art Museum.
AC 1945.494
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
https://www.pexels.com/photo/books-on-bookshelves-1166657/
https://pixabay.com/en/arrow-left-point-pointer-sharp-40169/
https://docs.oracle.com/javase/7/docs/api/java/awt/event/KeyListener.html
https://stackoverflow.com/questions/13216148/java-what-throws-an-ioexception

https://stackoverflow.com/questions/12700520/how-to-get-mouse-pointer-location-relative-to-frame
https://stackoverflow.com/questions/18356516/how-can-i-get-the-relative-and-absolute-cursor-position

Emily Potter-Ndiaye

December 7, 2018 at 4:34 PM

Details

EP

RE: Permission To Use Images For Escape Room

To: Peyton Lane 20, Cc: Stephen Fisher, David Little, Miloslava Hruba

Siri found new contact info in this email: Emily Potter-Ndiaye epotterndiaye@amherst.... add to Contacts...

Dear Peyton,
Thank you for your interest in the Mead's Rotherwas Room. You are permitted to use the photos you took of the Room itself, including the windows, but not including the contemporary art work by Yinka Shonibare (as he is a living artist and retains copyright to his work, that could get complicated). We do ask that you cite the artworks somewhere on your website, or in conjunction with it in any portfolios in the future. Here's the full information:
http://museums.fivecolleges.edu/detail.php?museum=ac&t=objects&type=all&f=&s=rotherwas&record=1
For these purposes you'd be fine to cite it as:
Rotherwas Room (17th century)
Mead Art Museum
AC 1945.494

I know I, and some of my colleagues in security are eager to see your finished project – let us know when it's ready to share! Good luck with final projects in the meantime.

All best,
Emily

Emily Potter-Ndiaye I she, her, hers
Head of Education and Andrew W. Mellon Curator of Academic Programs
Mead Art Museum I Amherst College
413-542-8229

Found in Sent - Exchange Mailbox

Peyton Lane 20 12:26 AM

Re: Permission To Use Images For Escape Room

To: Emily Potter-Ndiaye

Thanks so much! Our project is due tomorrow but it is totally ok! We found a way to avoid using the bookcase, but in the future would love to have permission to use it for any portfolios in the future, so if you could, please let us know if the artist grants permission.

Thanks again,
Peyton

See More from Emily Potter-Ndiaye

Emily Potter-Ndiaye 9:30 AM

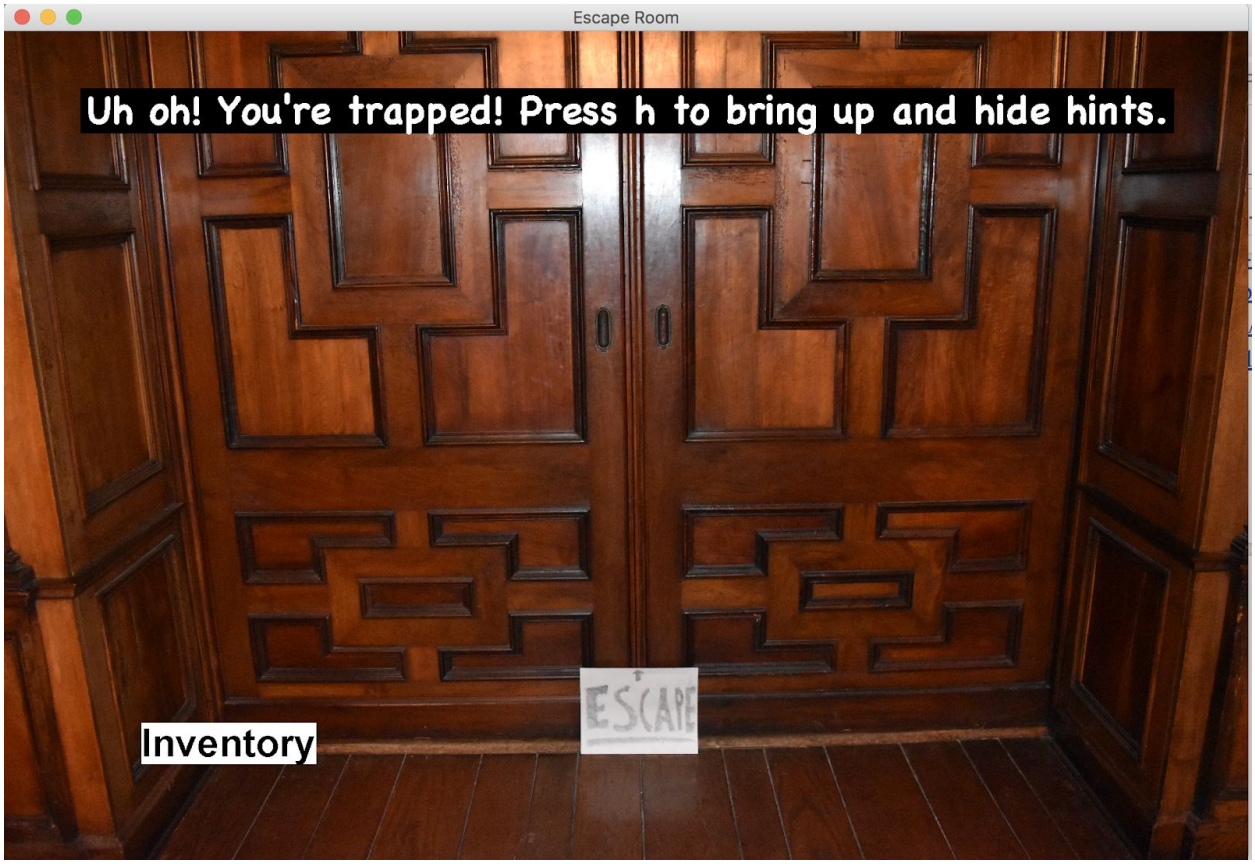
RE: Permission To Use Images For Escape Room

To: Peyton Lane 20

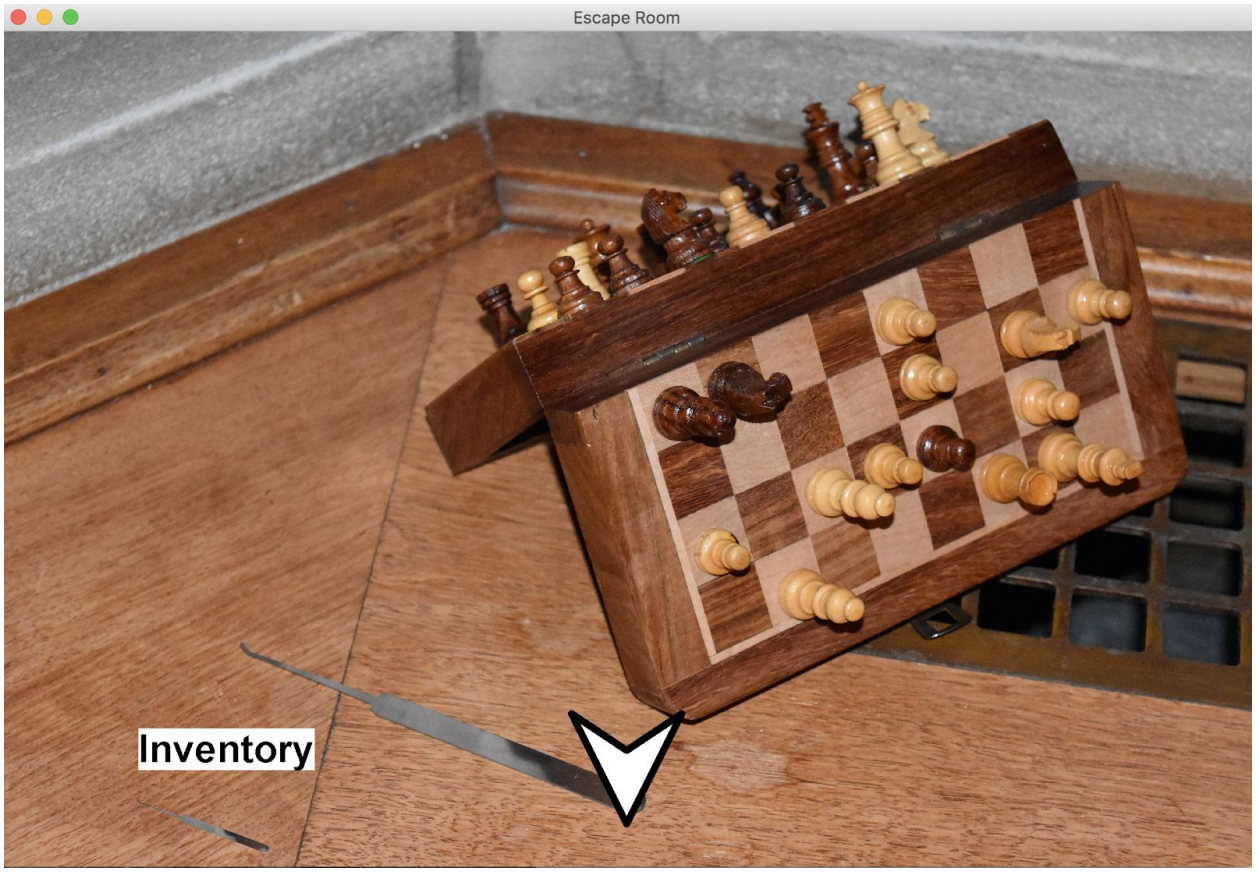
Hi Peyton,
We just got the okay from the artist's gallery for you to include the bookcase image in your game. Good luck finishing up the project!

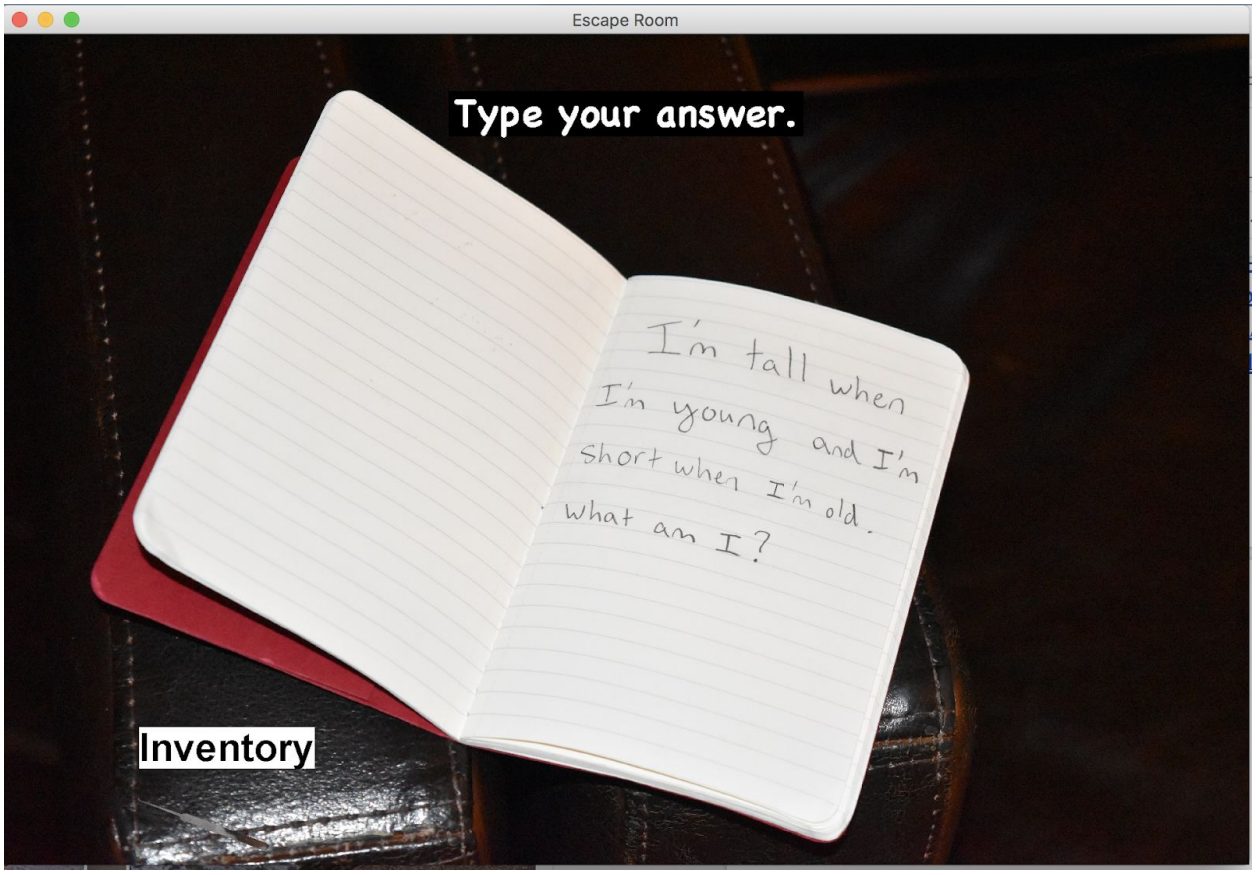
Emily

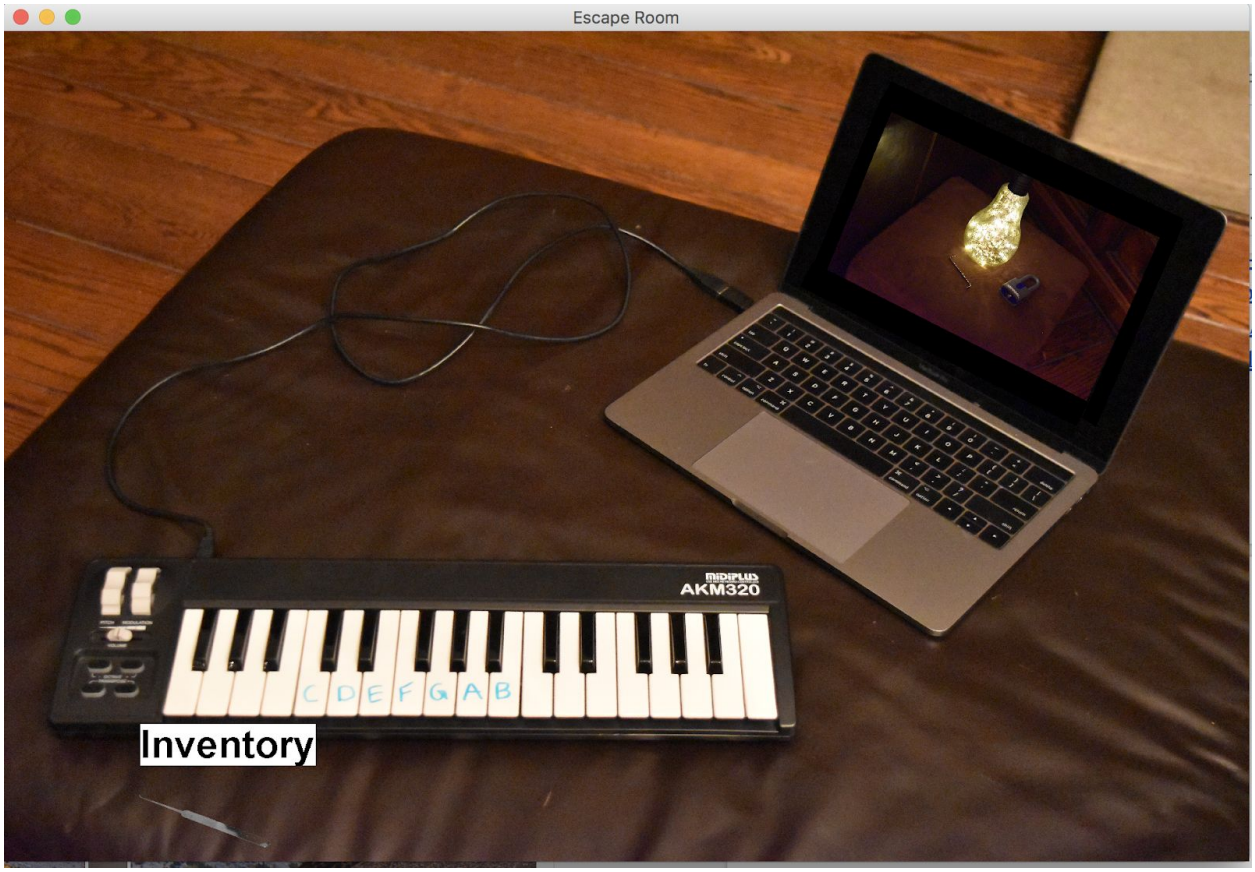
See More from Peyton Lane 20

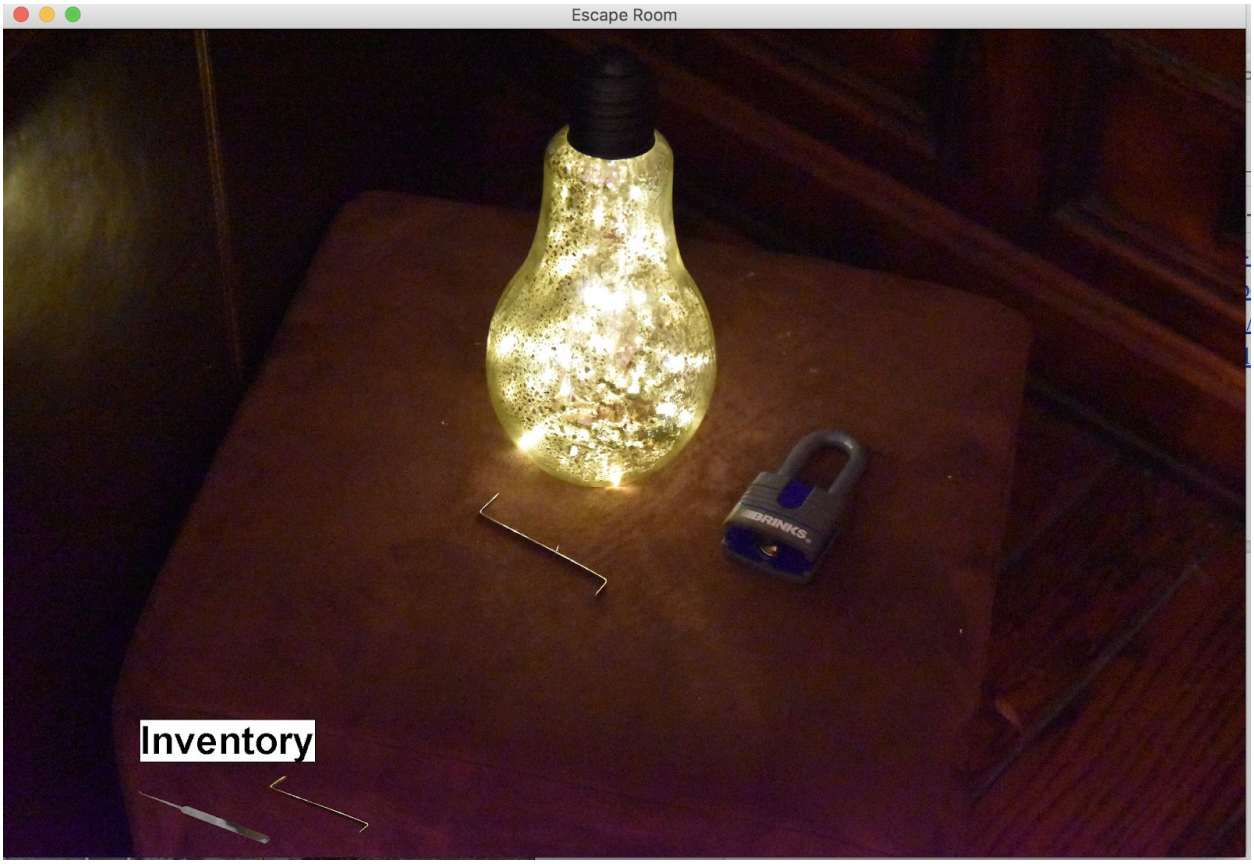

Escape Room
Uh oh! You're trapped! Press h to bring up and hide hints.
ESCAPE
Inventory


Inventory


Escape Room
Inventory


Escape Room
Type your answer.
I'm tall when
I'm young and I'm
short when I'm old.
What am I?
Inventory


Escape Room
AKM320
C D E F G A B
Inventory


Escape Room
BRINKS
Inventory


Escape Room
Congrats! You have escaped. Feel free to look around.
Inventory