Project Specification

Part 1: Generic Description

Our project is a dance-formation simulator. One of the most time-consuming aspects of dance performance is setting formations. This is a tool for a choreographer to use to help set formations. Generally, the choreographer knows what shapes they want to form, but putting people in certain places and figuring out who will go where in transitions is very time consuming. This simulator will have the choreographer input formations by clicking on the screen where they want the dancers to be. They also input additional information for their dancers, adding in names and choosing their representative color. They then select the next formations and can continue for as many formations as they wish.

The program takes in all of this information and displays the dancers from a top-down view, showing where each dancer should be at each formation, and also animates the full transitions between formations. In order to make sure people don't travel far, it will perform a regression algorithm to find the smallest cumulative distance between points. In doing so, it will save the choreographer and dancers lots of time.

Part 2: Class Descriptions

Classes: Animation, Button, Dancer, Display, Input, Main, Stage, and Welcome.
<u>Animation class:</u>

The Animation class extends Display, and is responsible for showing the dancers in formations on screen. This class also reads the txt file to create an Arraylist of Stages. This class calls methods in Stage to perform the regression analysis. The Arraylist of Stages is stored in Animation and used to display animations discretely as well as continuously through full animation.

Animation has the fields of int numButtonPresses, which counts the number of times the Next Formation button has been pressed, an int numberOfFormations, which stores the number of formations inputted. All fields are public because if another class wants to edit them it can. Other fields are ArrayList<Stage> initStages and stages, which store before and after regression analysis. It holds a Dancer[] animatedDancers, which are the dancers displayed on screen. It also has a JPanel field bottomPanel which holds the color key for dancers. It also holds a JLabel formationNumber which displays which formation is shown.

Animation has one constructor. It is called by the Input class. The constructor sets up the display with a topPanel with the formationNumber, JButtons for Next Formation, Full Animation, and Reset. It takes in an int argument and sets that as the numberOfFormations, another way Input and Animation communicate with each other.

Animation has several methods as well. readStates() reads the text file and loads the information into the ArrayList initStages and calls the Stage class regression method to load the information into ArrayList stages. It also has a paintComponent method that sets the background to white and loops through the animatedDancers[] to draw each dancer. It has an animate() method that calls animateTransition() for the numberOfFormations. animateTransition() animates the transition and calls nextAnimation(), which sets the new and next positions of the dancers, so it can animate the next transition. There is also a reset method that resets the dancers to the initial positions and resets counting parameters.

Button class:

The Button<T> class will be generic. It will work with the Input class and hold instances of the Dancer class. This is because the Input class uses a grid of generic buttons that hold instances of Dancers that the user is/wants to place and record. This easily and quickly connects Dancers with their coordinates in an interactive board of buttons. This class will contain the member field <T> obj, which is the element or instance of Dancer that the given button corresponds to.

The two constructors take one parameter of type String if the button has a specified named/labeled but does not hold a Dancer instance, and two parameters of type String and type T for if buttons with specified label/name and a specified object/Dancer instance.

This class contains 3 methods. There are to accessor methods, which are getType() to return the class/type that the Element of that button is an instance of, and getObj(), which returns the actual element instance that the button is linked to/storing. These methods are public so that other classes can access them to use them; this is necessary so that the Input class can get the Dancer information to check position occupation, send info to the Animation class, etc. The setObj(T o) is a mutator method that assigns an element of the generic type to the button. This is needed to store and place dancers on the board as the user specified. This is also public so that the Input class can move Dancers from one spot to another between formations and clear them after a formation is done.

Dancer class:

The Dancer class will describe each dancer, and hold all of the information for each specific dancer. There will be many instances of this class, depending on the number of dancers the user decides to input.

Each dancer will have private fields positionX, positionY, nextPositionX, nextPositionY, which are all doubles and holds positions and nextPositions. We don't want outside classes changing this; once position is decided/determined, it remains that way unless the class itself codes otherwise. All of these are stored as doubles, but getters and setters are ints because the input from the buttons are ints. They are stored as doubles so they can be edited in the update method which is used in the animation to make it smooth. Also, double fields of velocityX and velocityY hold the velocities. Each Dancer also has a String name and a Color color.

Dancer has multiple constructors, depending on information we have. One where it takes in ints for position, one that takes in two ints and a color (used in Input class), and one that takes in two ints, a color, and a String (used in Animation).

The methods for Dancer include mutator and accessor methods, getters and setters for each field. They all follow the form getVariable() and setVariable(newVariable). It also has an update method which updates the position to increment by the velocity/60. In place of a setter method for velocities, it has an updateVelocity method which sets the velocity to nextPosition-position. Dancers also all have a draw method which fills in a circle at at the positions scaled by the scale factors stored in Display. They have a toString method that returns their name and positionX and positionY.

Display class:

The Display class extends JPanel, and is an abstract class of what will be displayed, there are three subclasses, Welcome, Input, and Animation. This class holds constants, scale factors, and a field for the top panel, which is in both Input and Animation,

Each Display will have fields of int WIDTH, int LENGTH, dimensions of the panel. It will also hold doubles scaleX and scaleY that give the pixel/button ratio in each dimension so both Input and Animation can access the scale to input and draw the dancers. It also has a JPanel topPanel which is used in Animation and Input. All of these fields are static so they are the same in all of the child classes, so these fields can be accessed by every class.

Display has one constructor with no arguments that sets the preferred size to 1000,1000. This panel size will be the same in all of the instances of the child classes.

Display has two public static methods that can be accessed and used by another class. setScaleX(double x) and setScaleY(double y), which are called in Input once the number of button dimensions have been chosen. These are static because scaleX and scaleY are static, so setting them sets the value for every instance of Display.

Input class:

The input class is the part of the simulation where the user inputs and specifies the stage dimensions, dancers, and formations for the choreography. When the program is run, one instance of Input exists, as they are dealing with one choreography routine and group of dancers and stage at once. This class has only one constructor, which has a single parameter that is the JFrame holding it. There is only one because every input board will follow the same processes and contain the same things. All the user-specified things are also initialized and set in the say ways.

The member variables include two JFrames that store (and set up) the JFrames the Input panel is displayed on, JFrame inFrame, and the JFrame that Animation will be called into when the user finished inputting all the formations they want to. These frames are public because other files in the project are allowed to access them. There is an array of all the Button<Dancer>, which are placed in a grid and displayed. These represent the positions on the stage, which the user will click on and place dancers at for each formation in the choreography. These buttons are private and static, so that all the methods in that class can access and modify it throughout the input processes, which is needed to go through each case of each formation, but won't be modified or used in any other class. For the same reasons, the button dimensions variables, B_LENGTH and B_WIDTH, and the stage dimensions (in units of positions) horzPos and vertPos are all private and static. The button dimensions specify the size of each button according to the size of the panel; this is so that positions are scaled to the screen. Color variables store the colors used for non-occupied positions, Color BACKGROUND, placed Dancers (in any non-initial formation), Color PRESSED, and the color for the lines between buttons, Color lineColor. The background and occupied colors are constant throughout the simulation, and thus are final variables. They are all accessible/used throughout the Input portion of the simulation. Consequently, they are private (unused in other classes) and static (accessible throughout the entire class). lineColor was originally going to be adjustable, but this is a purely cosmetic and does not ever affect function of our program; thus the option to change it was cut; but this is why it doesn't need to be final. The thickness of the lines of the input grid are stored in public static int lineThick, which is defaulted to 2. This is public because it does not need to be private to only this class, and is static so that it can be accessed and adjusted within the simulation. All the variables that are public and static are accessed throughout the Input portion of the simulation. These variables include the JButton bScatter, which provides the option for a random placement of dancers for a formation; the counters, int fNum and int dNum and int dMax, which keep track of the number of formations that have been inputted, the number of dancers that have been placed so far in the given formation and the total number of dancers in the routine; the txt File writing variables, File colors,dancers,formations, which respectively are written to hold the dancer colors, dancer names, and coordinates, with the first two files only being written to for the initial formation and the formations file containing the coordinates of dancers from each formation; each of the files have a bufferedwriter, BufferedWriter cb,db,fb, respectively, used write the necessary information to the files; a Random variable, Random rand = new Random(), that is used when bScatter is clicked to place dancers at random positions on the stage for that formation. The private int variables, tmpV and tmpH, are temporary variables that hold the vertical and horizontal coordinates of the position/button clicked on the board. They are used within the actionPerformed code, and are checked each time to make sure they are valid positions on the stage. The boolean firstPick is a flag of check for whether the initial formation is being set or if that has already been done. Once it has, it is set to false and so all actions and code for it are no longer gone through. This is because once the user specified all the specific dancers and the colors and names, this information is not displayed/determined until the Animation and thus is not displayed.

There are many methods in this class. The public void methods that follow are public and void because they set up components of the class and do not need to return/provide any information outside of what they do in their code. The first method is setChoices(), which goes through the process of prompting the user for the necessary information to set up the board of buttons to simulate their stage and know how many dancers they have in their choreography. setFileStuff() resets/clears and initializes the variables necessary for producing the txt files to hold the dancer, color and coordinate information (sent to Animation class). initButton() initializes the Button<Dancer>[][] board/array of buttons that the user interacts and inputs information with. actionPerformed() is invoked when a button is clicked, producing an ActionEvent action. When button containing a dancer is clicked, this establishes what will occur; it goes to the initialFormation placement processes or the nextFormation placement processes, depending on the boolean firstPick. hasDancer(int w, int l) returns true if there is a dancer at the inputted coordinates (row,col); false if that position is unoccupied. initialFormation() is the process (called in actionPerformed) for setting the initial formation as well as the dancer and color information. nextFormation() is the action (called in actionPerformed) for handling non-initial formations. toTxt(int w, int l) stores the coordinates of an added position/dancer at the inputted coordinates. In the txt files from the initial formation, all three files are written in. after the initial, only coordinates. addDancer adds a dancer to the stage at the desired location coordinates. checkdMaxReset() does nothing unless the max number of dancers have been placed in a given formation. If max number has been placed, then will go through process to transition to next formation or end if at least 2 formations have occurred and the user wants to finish the choreography. All of these methods were public void.

The last method, topPanel(), sets up the top panel, a returns a JPanel containing the buttons and sliders for the user to adjust the visual format/display preferences. This top panel is then added to the JFrame inFrame above the input board.

Main class:

The main class will be public and with only one class in its file. Main contains a public static void main(String[] args) method which will be used to instantiate and initialize the JFrame format and contents to hold the Welcome screen/panel. This will start the simulation sequence (which are all instantiated and started in order by one another).

Stage class:

The Stage class holds an array of Dancers and holds most of the code for the regression analysis to figure out which dancers go where. The algorithm takes in an array of dancers in the next position, and finds the all permutations of the dancers, which is why the code runs slowly with a lot of dancers. It stores all of these permutations in an ArrayList and also finds the cumulative distance that each dancer would have to travel for each permutation and stores it in an array. Then, using that array, we found the permutation with the least cumulative distance and set those positions for the next set of dancers.

The Stage class has all public fields so they can be accessed from Animation. It has fields of int WIDTH and LENGTH, which store dimensions of the stage. It has a Dancer[] called dancers, which holds all of the information for the dancers. It also has an int field cumulativeDistCount and a double[] called cumulativeDist. These are used in calculating the regression.

Stage has several fields, all public because it needs to be accessed by Animation. It has a public static int cumulativeDistCount, and a double array cumulativeDist[].

The Stage class has two constructors, used in Input and Animation, depending on the data available. There is one constructor that takes in two ints for dimensions and one constructor that takes in two ints and a dancer array.

Stage methods are where the regression takes place. The public method nextFormation(Dancer[] nextDancers) returns a Dancer[] of the dancers in Stage in the positions of the dancers. This method calls recursive method factorial, which finds the number of permutations of the dancers. It also calls recursive method permute, which takes in the dancer array and finds all of the permutations and stores them in an ArrayList. Permute uses the method swap, which switches two elements in an array. Stage also has a method findCumulativeDist which returns a double of the sum of all of the distances of a dancer to its next position, and is also called in nextFormation.

Welcome class:

The Welcome class extends Display, so it is a JPanel that shows the welcome screen when you first open the program. There's one instance of this each time the main method is called. It holds JPanels, JLabels, and JButtons. There is an Instructions button that displays instructions when you click it. The Welcome class also has a button that will instantiate an Input, starting the simulation.

Welcome class doesn't have any fields, the majority of the setup is in the constructor. It only has one constructor that does not take any arguments. It is public so it can be called by the main method. The constructor calls method setInstructionsPanel and setBeginPanel. setInstructionsPanel sets the instructions panel to a 3x1 GridLayout, adds the JLabel for the title, adds a Button for Instructions, and adds a beginPanel to the 3rd grid spot. setBeginPanel adds button to beginPanel that when clicked, will begin the simulation.

Part 3: Figures

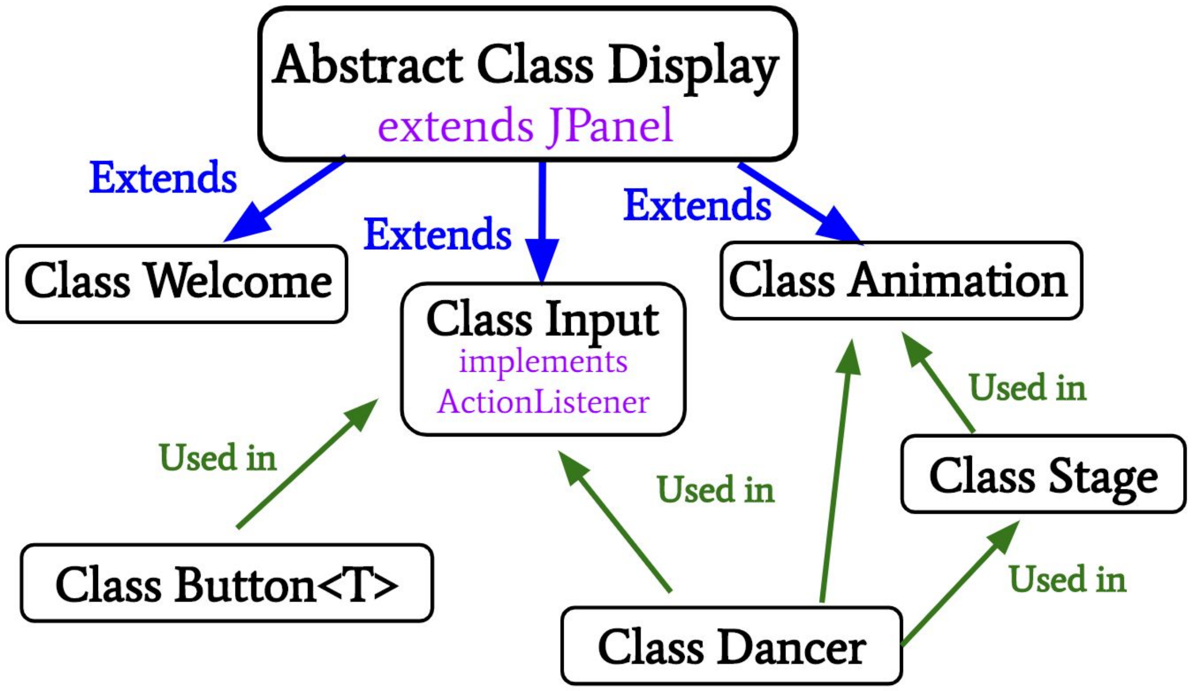

FIGURE 1: Diagram of Class Relationships

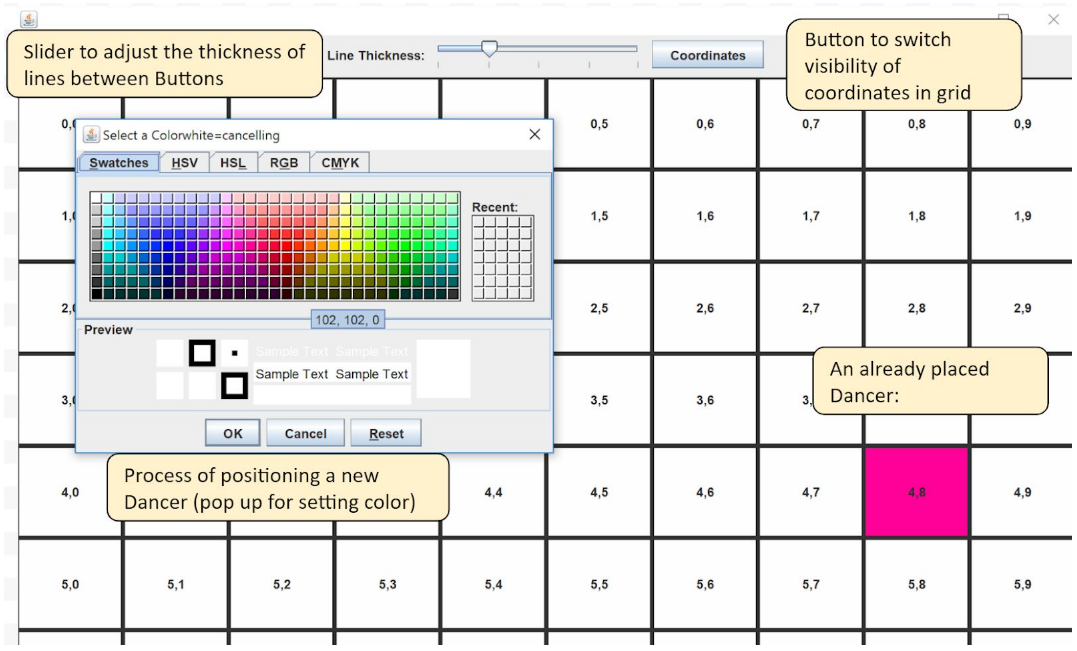
Slider to adjust the thickness of lines between Buttons
Line Thickness:
Coordinates
Button to switch visibility of coordinates in grid
Select a Colorwhite=cancelling
Swatches
HSV
HSL
RGB
CMYK
Recent:
102, 102, 0
Preview
Sample Text Sample Text
OK
Cancel
Reset
An already placed Dancer:
Process of positioning a new Dancer (pop up for setting color)
0,5
0,6
0,7
0,8
0,9
1,5
1,6
1,7
1,8
1,9
2,5
2,6
2,7
2,8
2,9
3,5
3,6
4,0
4,4
4,5
4,6
4,7
4,8
4,9
5,0
5,1
5,2
5,3
5,4
5,5
5,6
5,7
5,8
5,9

FIGURE 2: Input class → Initial Formation, placing dancers at positions and inputting their information (color and name)

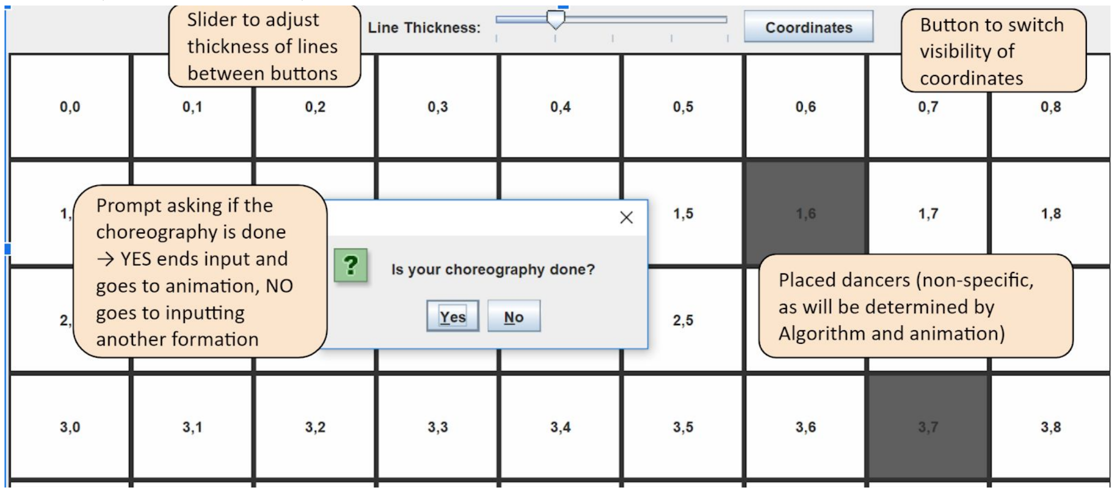
Slider to adjust thickness of lines between buttons
Line Thickness:
Coordinates
Button to switch visibility of coordinates
0,0
0,1
0,2
0,3
0,4
0,5
0,6
0,7
0,8
Prompt asking if the choreography is done → YES ends input and goes to animation, NO goes to inputting another formation
Is your choreography done?
Yes
No
1,5
1,6
1,7
1,8
2,5
Placed dancers (non-specific, as will be determined by Algorithm and animation)
3,0
3,1
3,2
3,3
3,4
3,5
3,6
3,7
3,8

FIGURE 3: Input class → next Formation, identifying where non-specific dancer placements will be fore each non-initial formation.

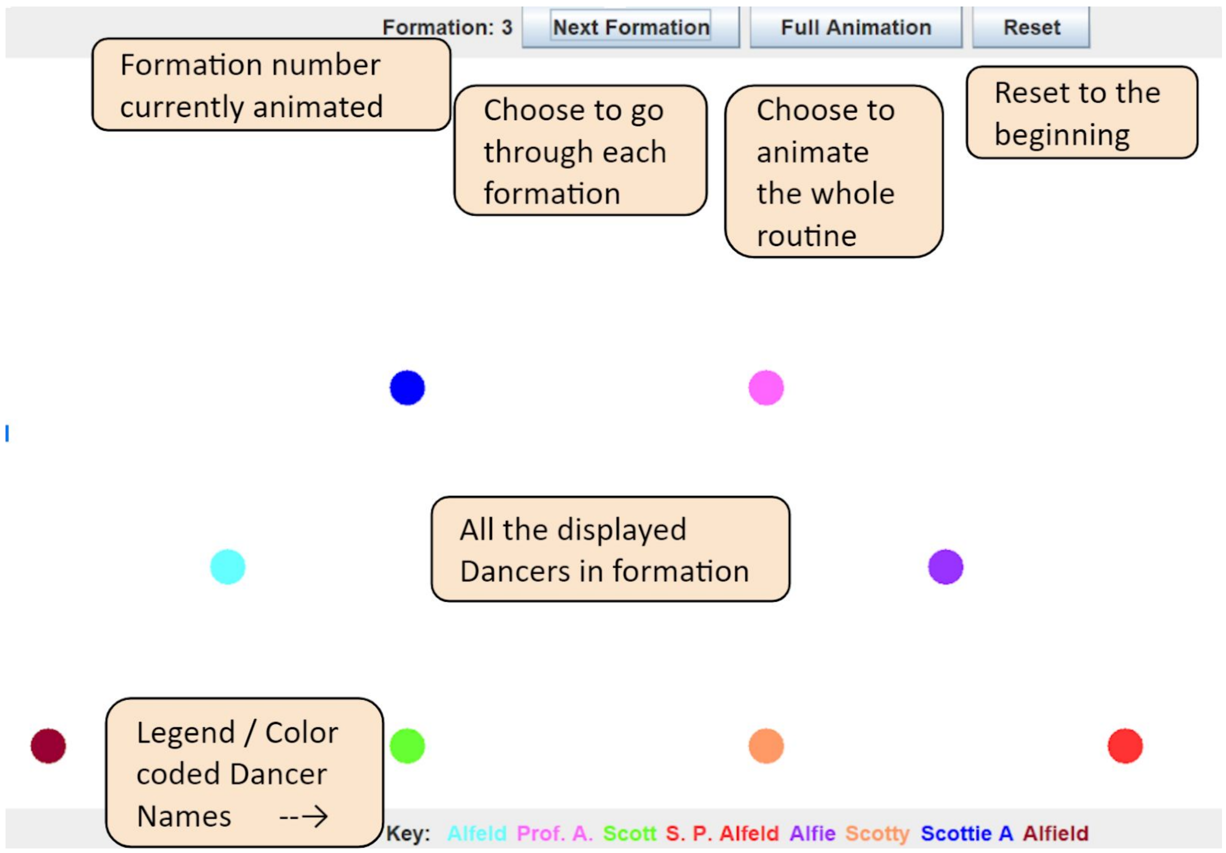
Formation: 3
Next Formation
Full Animation
Reset
Formation number currently animated
Choose to go through each formation
Choose to animate the whole routine
Reset to the beginning
All the displayed Dancers in formation
Legend / Color coded Dancer Names --→
Key: Alfeld Prof. A. Scott S. P. Alfeld Alfie Scotty Scottie A Alfield

FIGURE 4: Animation class → Animation of formations