**Stack-based Push-Pop 2D Transformation Matrix**

|  |
| --- |
| Henry Nguyen  [henry\_nguyen777@hotmail.com](mailto:henry_nguyen777@hotmail.com) |

**ABSTRACT**

February 18 2018. This paper describes my approach to the Stack-based Push-Pop 2D Transformation Matrix, including related research references. The paper identifies and addresses the six main goals of the project: (1) Use of Java’s Generic ArrayList for a graphics application and an array; (2) design class files to build multiple parent-child hierarchical objects; (3) design and apply push and pop stack structure to properly associate hierarchical objects to build gestures for 2D transformation; (4) design a custom function/method blocks to add a unique movement of parent-child based objects; (5) add both mouse and keyboard interactions to transform hierarchy chained objects expressing different motion; (6) incorporating randomness; and (7) documenting my design approach/process and research endeavor.

**INTRODUCTION**

My project tries to reflect real life objects, including a human and dog, and things of nature, i.e., weather conditions and trees. The following content in this paper addresses how I approached each of the requirements and my research endeavors.

**ARRAYLIST and ARRAYS**

**Trees**

Originally, I planned to use the ArrayList portion of the project for rain drop objects, but after research and following tutorials online, I decided I will use it for trees instead. First, I declared the ArrayList and opened a new tab for the Tree class. The trees are added using the add() function which passes values based on the cursor’s x-axis and y-axis. This allows the tree to be created on where the user chooses, and how tall they want to tree to be. Compared to other objects, such as human, the Tree class is accessed using a forloop and a get() function, which is inside the draw() function. It would get the index and pass it into the get() function, which then gets and calls the display() function, which draws the trees. The display() function consists of drawing the wood of the tree using rect() function, and the leaves are drawn using the triangle() function as the base and then add on top of it using more triangle() functions with a forloop and changed values. It also has a keyReleased() function in the code block Weather() (which is discussed later in the paper) to prevent the user from spamming trees.

**Snow**

The Snow class uses an array, which is initialized and populated in a function called getSnow(). I referenced one of the Processing tutorials on arrays that shows how to make spots on a screen (Processing, n.d.). Since it was a simple example and consistent with lack of randomness, I did not make much changes to the code other than values (such as number of snow and y-axis positions to make the snow seem more natural), variable names, and encapsulation.

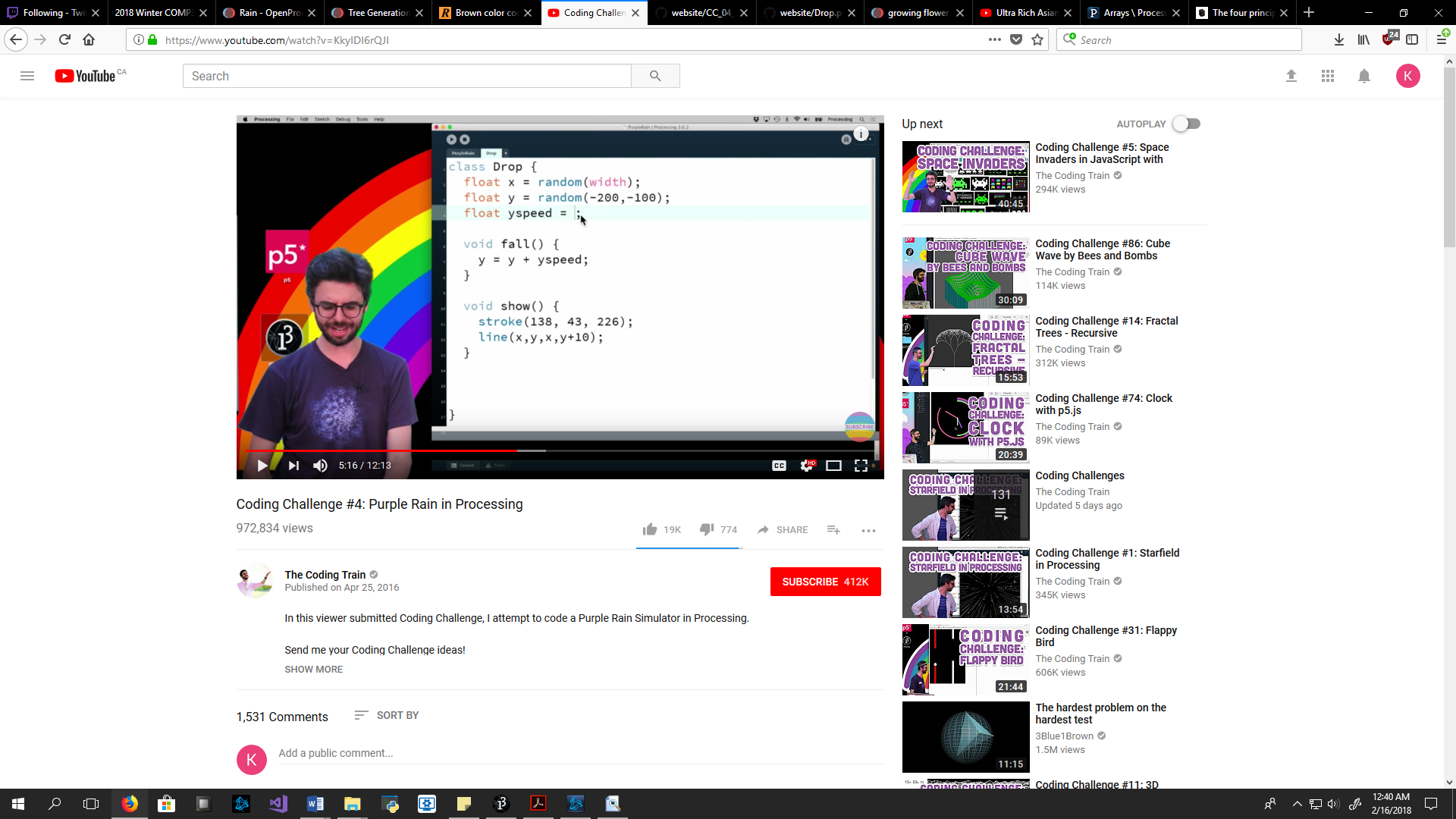


**Figure 1. Processing array tutorial example for “spots”. Image: Spots by Processing**

The figure above is the one provided in the example code, however compared to my snow, it does not change direction, rather its’ y-axis position is reset and moved off screen.

**Rain and Blizzard**

The Rain and Blizzard arrays are similar with the only difference being that: (1) the Blizzard class uses rotation() function to make it appear as the snow is travelling in high wind; (2) uses ellipse() to create the shape of snow; (3) no randomness to the size of the snowfall; (4) uses push and pop matrix to prevent the random rotation from affecting other objects; and (5) that it whitens the screen in the generateBlizzard() function to make it seem more natural. The reference to these arrays are from a coding challenge provided by a broadcaster named The Coding Train (The Coding Train, 2016).



**Figure 2. The YouTube series tackles various creative coding challenges in Processing. Image: The Coding Train on YouTube.**

The problems faced when incorporating rain into my project was that the tutorial used lines() functions, stroke() and strokeWeight(); however, since the lines() function cannot be used with the fill() function, the color couldn’t be changed unless we use the stroke() function, which would change *all* line colors in the project. So I used the rect() and fill() function to draw the rain drops, changed values on the length and thickness variables to smaller values, and y-axis position to a higher negative value to simulate real rain. As for the Blizzard class, I played with it and referenced the Rain class rather than use the online video.

**OTHER CLASS FILES AND MULTIPLE PARENT-CHILD HIERARCHICAL OBJECTS**

Above I talked about the ArrayList and array classes, so I won’t go into as much detail on those classes in this section.

**Human**

The human was first built and was referenced from the 2D transform example provided to us (Processing, n.d.). It was a way of testing classes and transformation; some values changed include different shape sizes, colors, and interactable legs. If the human object steps outside or inside the house, then they lose or gain 20 pixels on their y-axis value to make it seem like they’re grounded.

****

**Figure 3. 2D Transform Example. Image: Slide 1.**

**House**

The house has no reference since it only requires common sense. I started with the foundation and built “upwards”, including the walls and roof. If the house didn’t look “right” then I would reorder it or add more objects to the House class. The doors are interactable if the Human object reaches the left or right door, causing them to open and close.

**Dog**

The Dog class is also drawn without reference, but to make it more interesting and to fill the mouse interaction requirement, I decided to add the increaseSpeed() function which allows the user to click on the dog and increase his tail speed (until a limit is hit then it resets). The Dog class functions are referenced to the Human class since it was made right after and for practice.

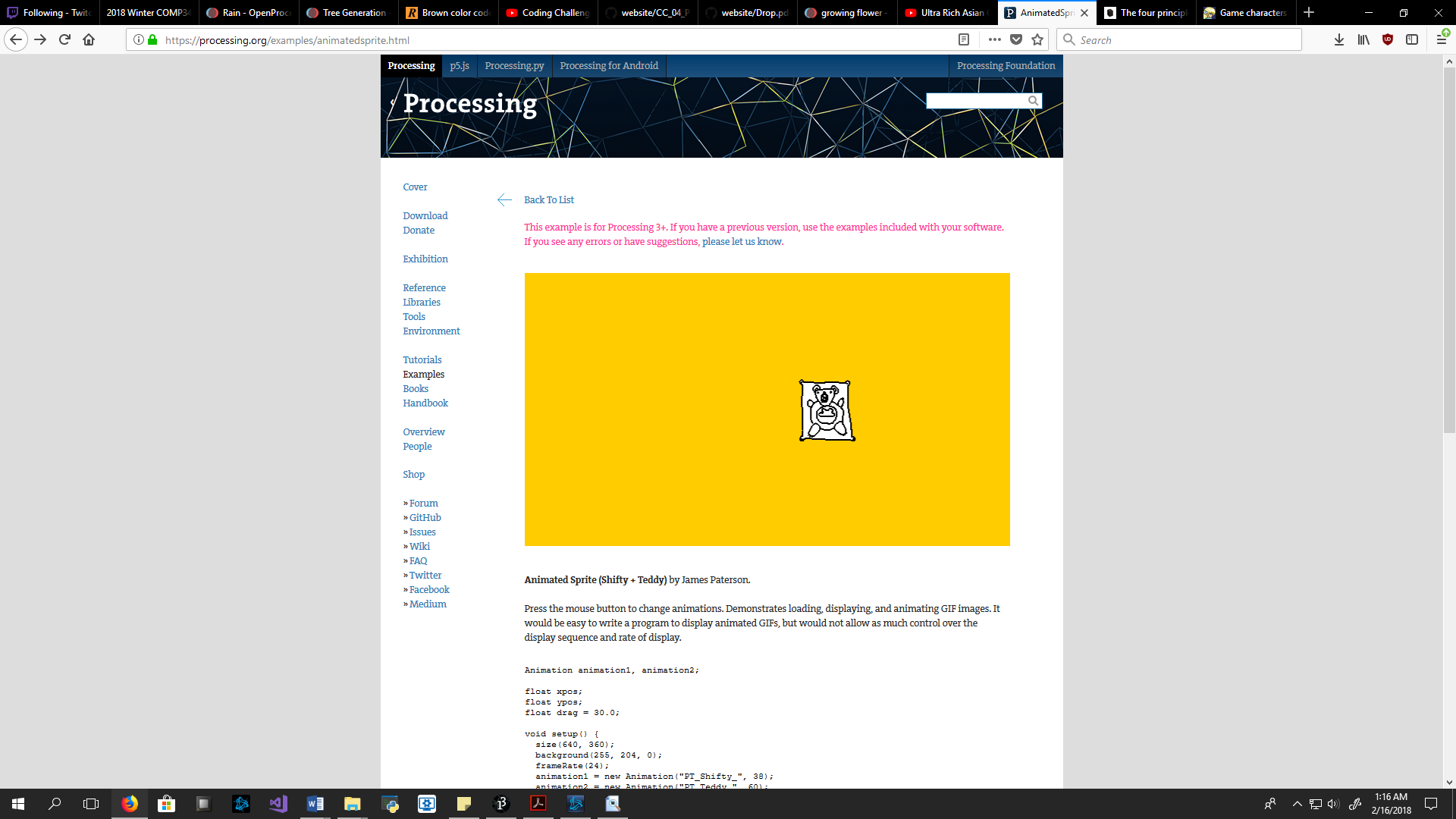
**Bird**

Originally, the Bird class was drawn and used atan2(), sin(), and cos() functions to determine how it would follow the mouse cursor; however, the bird was difficult to manually draw and the math functions were hard to deal with. To deal with the issue of drawing the bird, I decided to use a sprite, which I got from an open source project (with no form of copyright or licenses) by bevouliin.com (bevouliin, 2014).



**Figure 4. Birds Attack Sprite Sheets. Image: bevouliin from Open Game Art.**

I then, individually, resized the sprites, cut them out, and placed them on transparent backgrounds using GIMP (GIMP, n.d.). To address the ability for the bird to follow the mouse cursor, I used Processing’s tutorial on animation sprites. (Paterson, n.d.)



**Figure 5. Animated Sprite of Teddy Bear. Image: James Paterson from Processing.**

I then added key interaction, altered retrieving filenames for .png files, y-axis, and differential y-axis.

**Incorporating Randomness**

The random() function is used within the Blizzard, Rain, and Tree classes. Blizzard uses a rotate() function that accepts a random integer between 20 and 100 for the snow. The Rain uses a map() function that accepts a random value between 0 and 10 to determine the thickness of the rain drops. The Tree has 4 random variables for tree width and height, and the leaves’ width and level (layers of leaves).

**Flags**

I’ve added flags since I wanted the objects to stay on screen rather than disappear once a key is changed, or absence of mouse click. To do this, I added Boolean values for each object that is not initially there on screen and has to be called in, for example, snowFlag. I placed these flags in the Weather() function to keep it clean and if the user wants an object, they would press a key and the if-condition would return the flag as true. When the flag is true, then the checkFlags() function is called after the Weather() function, which checks if the flag is true, if so, then execute the function, for example, generateSnow(). To remove these flags, I put an if-condition for the key, ‘6’, which turns all flag values to false.

**Conclusion**

I learned a lot from this project and considering this is my first-time programming in something close to Java, I believe I made good progress. I managed to address every goal of the project, such as using an ArrayList and array for trees and weather conditions; design class files to build parent-child hierarchical objects like Human and Dog; apply the push and pop stack structure such as Blizzard and House; design custom function/method blocks like generateSnow() and Weather(); added both mouse interaction and keyboard interactions expressing motion, such as clicking the Dog to change tail wag speed and pressing 3 to create snowfall; and added randomness to some array and ArrayList classes.

# **References**

bevouliin. (2014, December 23). *Game Characters - Flying Bird Attack Sprite Sheets*. Retrieved from Open Game Art: https://opengameart.org/content/game-characters-flying-birds-attack-sprite-sheets

GIMP. (n.d.). *GIMP*. Retrieved from GIMP: https://www.gimp.org/

Paterson, J. (n.d.). *Animated Sprite (Shifty + Teddy)*. Retrieved from Processing: https://processing.org/examples/animatedsprite.html

Processing. (n.d.). *2D Transformation*. Retrieved from Processing: https://www.processing.org/tutorials/transform2d/

Processing. (n.d.). *Arrays*. Retrieved from Processing: https://processing.org/tutorials/arrays/

The Coding Train. (2016, April 25). *Coding Challenge #4: Purple Rain in Processing*. Retrieved from Youtube: https://www.youtube.com/watch?v=KkyIDI6rQJI