NEW YORK INSTITUTE OF TECHNOLOGY

Vancouver

CSCI 507 Data Structures - Spring 2024 VA1

Final Project Report

Data Structure Visualizer

Submitted by:

Team Lucky

Sanjay desu: 1343021 | jdesu@nyit.edu

Chuan-En Hou*: *1339293 | advance.houdavid@gmail.com

MS Cyber Security, New York Institute of Technology, Vancouver Campus

Table of Content

Introduction

Project Overview

This project is a java-based graphical user interface (GUI), showcasing the animation of operations of different data structures, such as addition, deletion and so on. The motivation of doing this project is that data structures are important. They are the backbone of all modern tech. Simple devices like old-school MP3 players to complex autopilot systems, they all rely on the data structures.

Since data structures are so important, it is crucial for us to understand this subject. However, data structures can be challenging to understand due to their abstract nature. Without physical forms, people usually struggle to understand how they actually work.

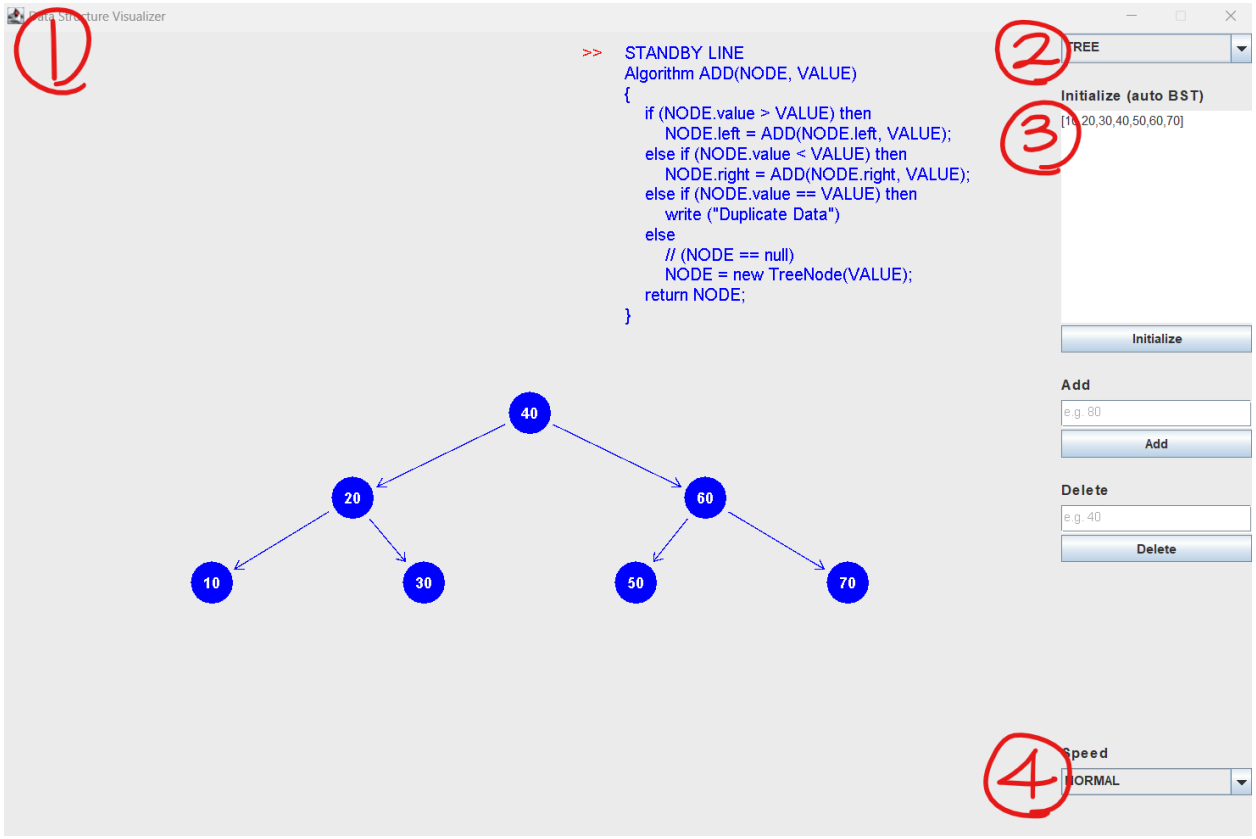
In order to address the problem, we decided to do this project - a Data Structure Visualizer (DSV). This application is designed to make data structures easy to understand by using graphical components to demonstrate how they work. Our DSV could serve as an effective learning tool and enhance the learning experience.

For the source code and user manual, please refer to the appendices section for more details.

Graphical Design

Layout

The layout can be divided into four sections, as shown in the following image:

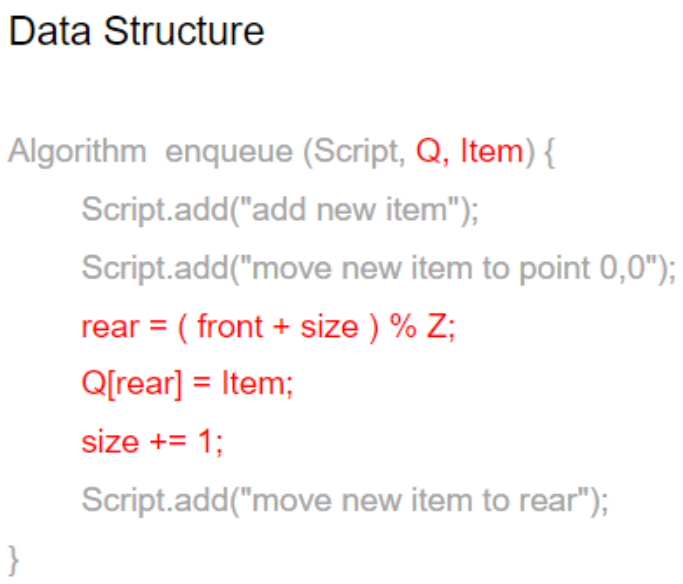


1. Animation Section (Left-hand side): This is where all the animations take place. It can be further divided into two parts:
    a. Graphical Part: This part runs animations by moving arrows, circles and squares.
    b. Pseudo-code Part: This part indicates where the code is currently executing.

2. Data Structure Dropdown List (Top right corner): Users can choose which type of data structure they want to see.
3. Operational Area (Right-hand side): Users can interact with data structures here, including initialization, insertion, deletion, and more.
4. Speed Control Dropdown List (Bottom right): Users can adjust the animation speed based on their preferences.

Animation

Since a data structure is an abstract concept, its entire operations are invisible. Like the red color code in the following image:



To make it visible to users, we study how old-style cartoons are made.

How old-style cartoons are made?

1. **Scriptwriting**: A storyteller writes a script that records all the details, including character movements such as walking and kicking.
2. **Animation Production**: The script is then handed over to Disney. There, each character is drawn onto the roll film, capturing their actions frame by frame.
3. **Playback**: When the roll film is played, it comes to life as a vivid cartoon with motion.

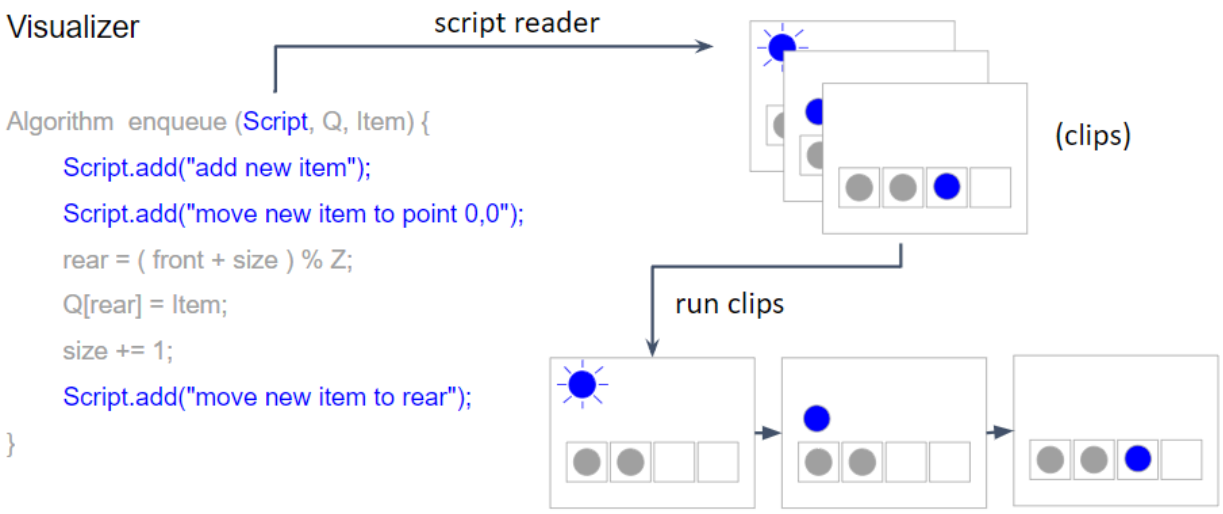
Similarly, we can create our animation using the same process as making an old-style cartoon

How was the DSV animation made?

1. **Scriptwriting**: We are the storytellers. We write a script that records all the details, including data node's movements, such as moving a node from point A to point B.

2. **Animation Production**: The script is handed over to a script reader. Here, each data node is drawn onto the clips, capturing their movements clip by clip.
3. **Playback**: When a movie (or a collection of clips) are played, they come to life as a vivid animation.

The overall creation of a DSV animation is shown in the following image:



By mimicking the way old-style cartoons are produced, we are able to create the animation for the DSV, thereby bringing the operations of data structures to the front for users.

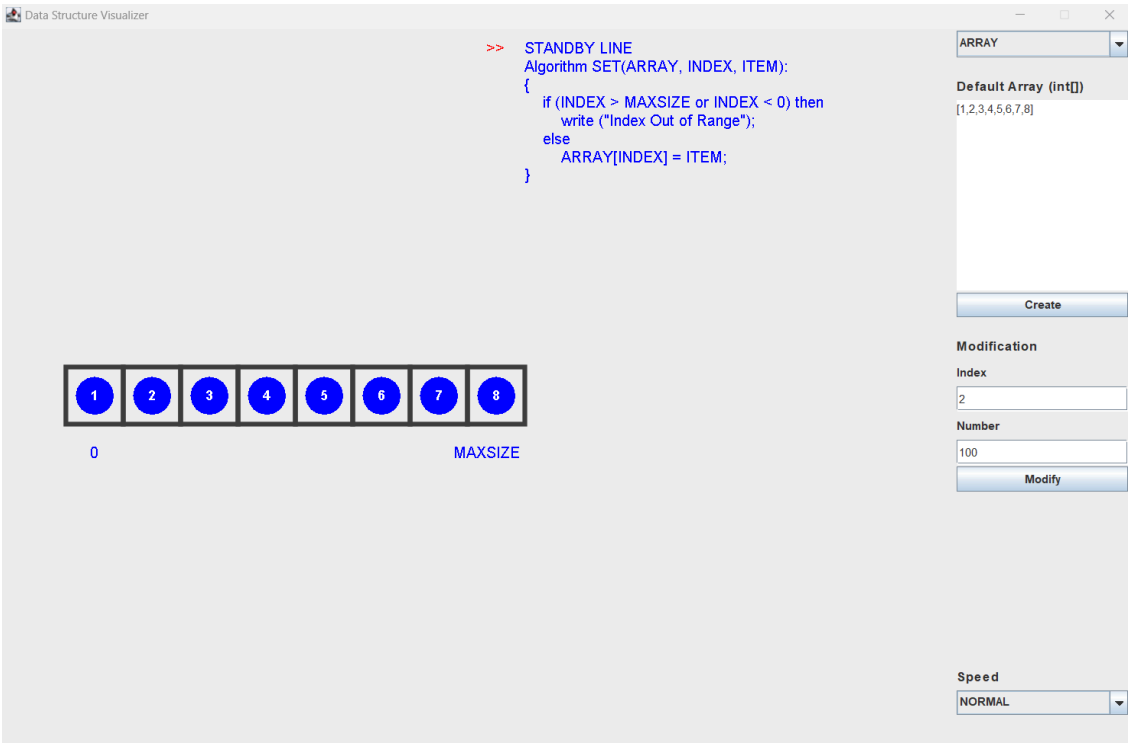
Page

Array

For the array page, two functions are provided.

- Initialization
    - This function is used to initialize an array. Please enter the size of an array and press the “Create” button.

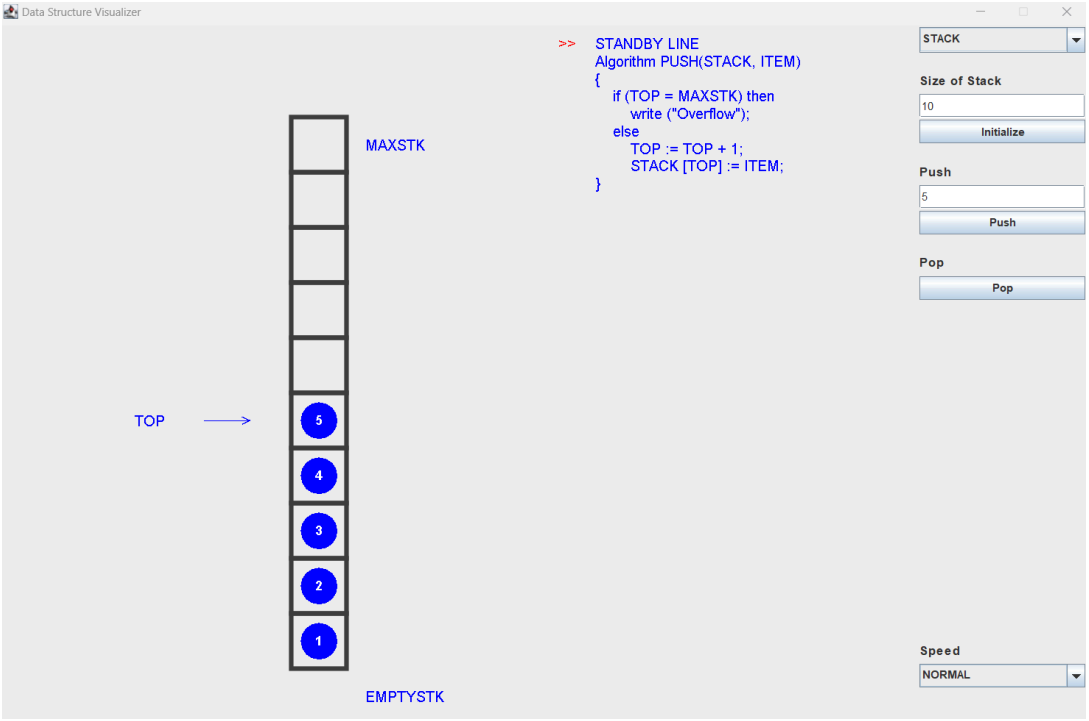
      Please note that due to the limitation of screen size, the maximum capacity of an array is 10.
- Modification
    - Users can modify the data from an existing array by providing the index and the number they want to replace.



Stack

For the stack page, three functions are provided:

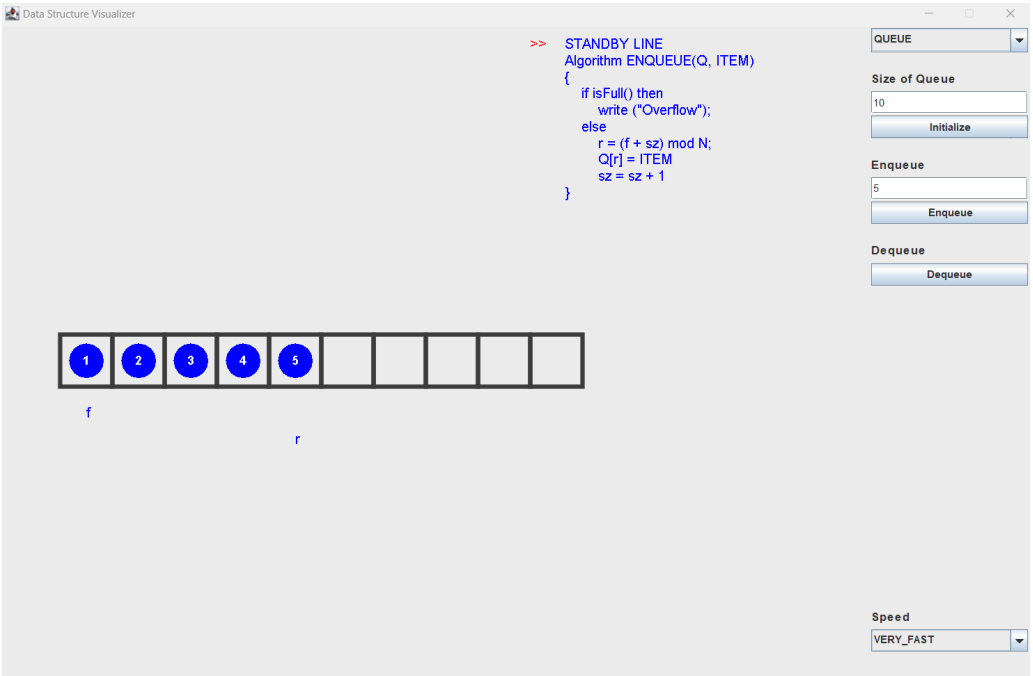
- Initialization
    - This function is used to initialize a stack. Please enter the size of a stack and press the “Initialize” button. An empty stack with that size will be created. Please note that due to the size of the screen, the maximum size of a stack is constrained to 10.
- Push
    - This function is used to push an item into a stack. Please type in a number and press the “Push” button to perform the push operation.
- Pop
    - This function is used to pop an item out from a stack. Please press the “Pop” button to perform the pop operation.



Queue

For the queue page, three functions are provided.

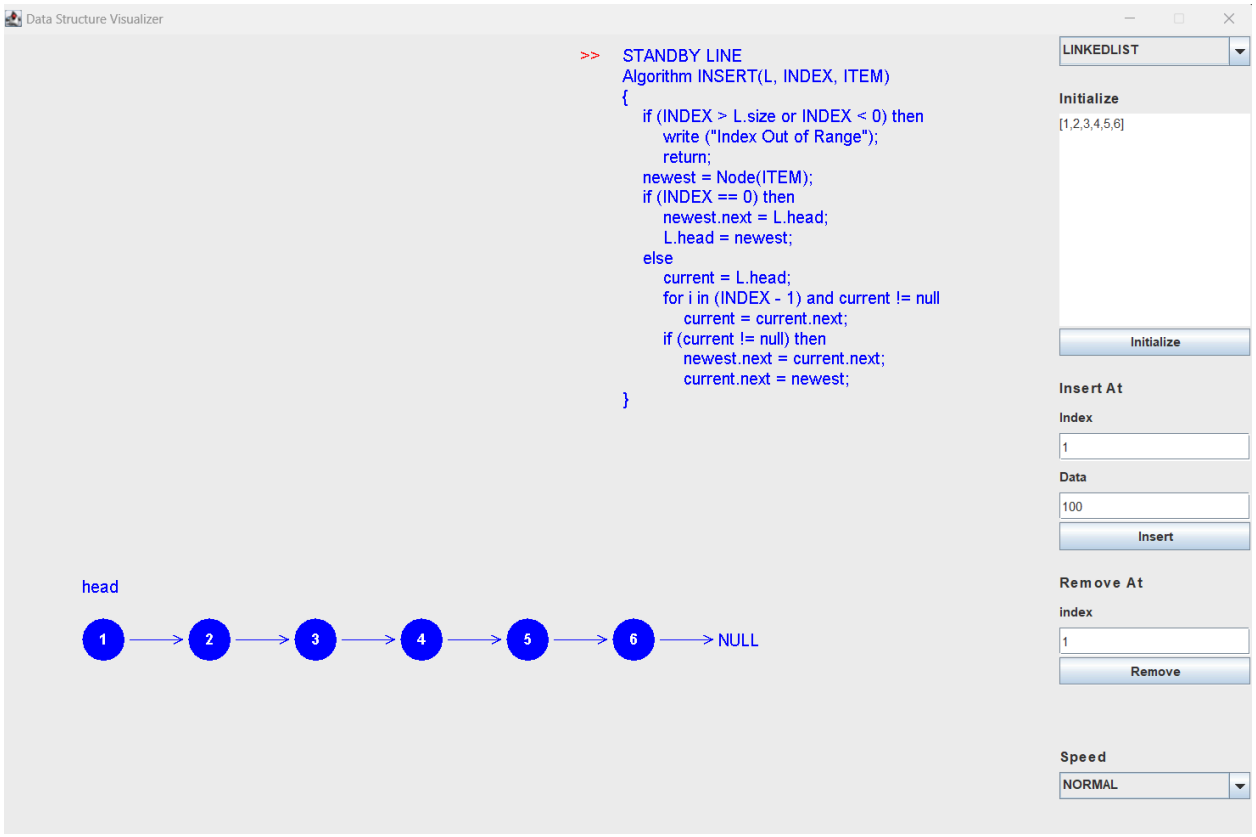
- Initialization
    - This function is used to initialize a queue. Please enter the size of a queue and press the “Initialize” button. An empty queue with that size will be created. Note that due to the size of the screen, the maximum size of a queue is constrained to 10.
- Enqueue
    - This function is used to enqueue an item into a queue. Please type in a number and press the “Enqueue” button to perform the enqueue operation.
- Dequeue
    - This function is used to dequeue an item out of a queue. Please press the “Dequeue” button to perform the dequeue operation.



Linked List

For the linked list page, three functions are provided.

- Initialization
    - This function is used to initialize a linked list. Please enter a complete set of data inside a pair of brackets, such as [1,2,3,4,5]. And then press the “Initialize” button to create a linked list. Note that due to the size of the screen, the maximum size of a linked list is constrained to 7.
- Insert At
    - This function is used to insert an item at a designated index in the linked list. Please assign an index value and a data value to the index field and data field respectively. Press the “Insert” button to perform the insert operation.
- Remove At
    - This function is used to remove an item from a designated index in the linked list. Please provide the index of an item you want to remove. Press the “Remove” button to perform the remove operation.



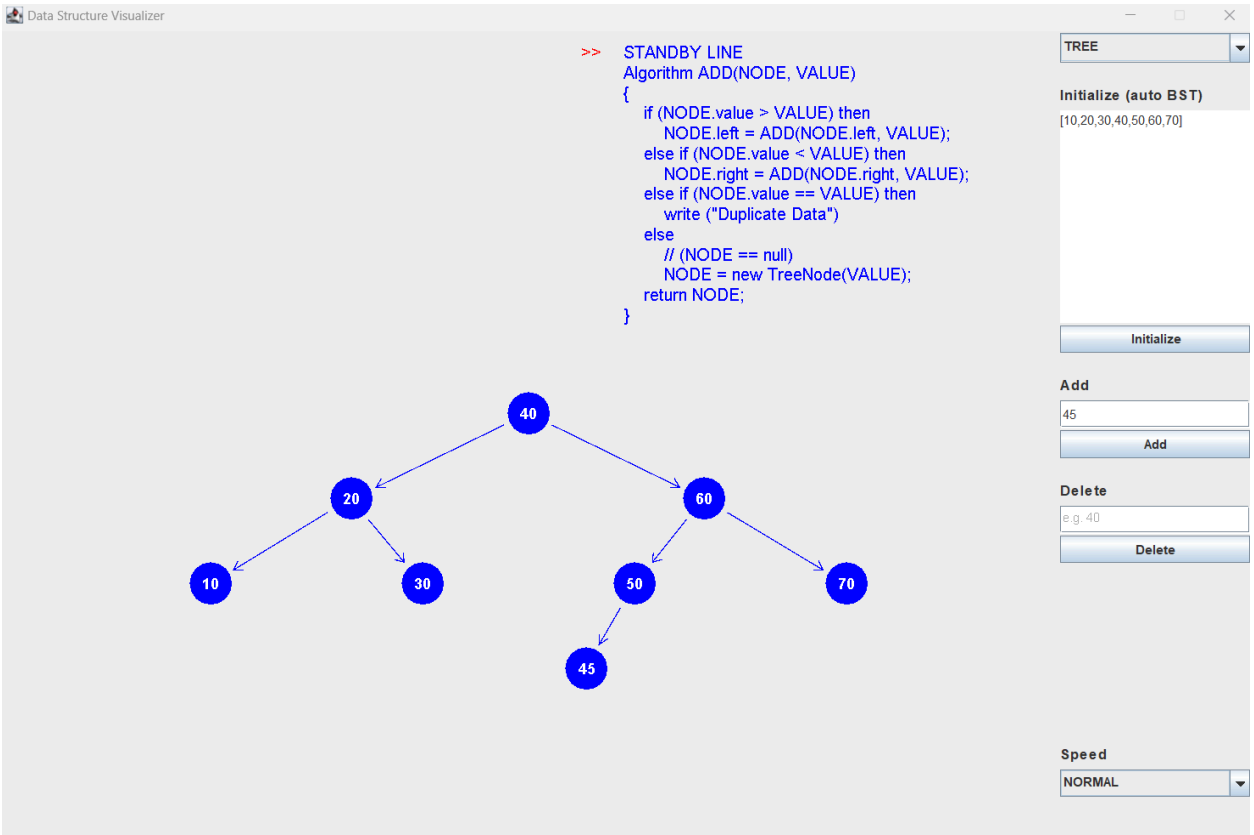
Binary Search Tree

For the binary search tree, three functions are provided.

- Initialization
    - This function is used to initialize a binary search tree. Please enter a complete set of data inside a pair of brackets, such as [10,20,30,40,50]. And then press the “Initialize” button to create a binary tree. The system will automatically create a binary tree out of the data set.

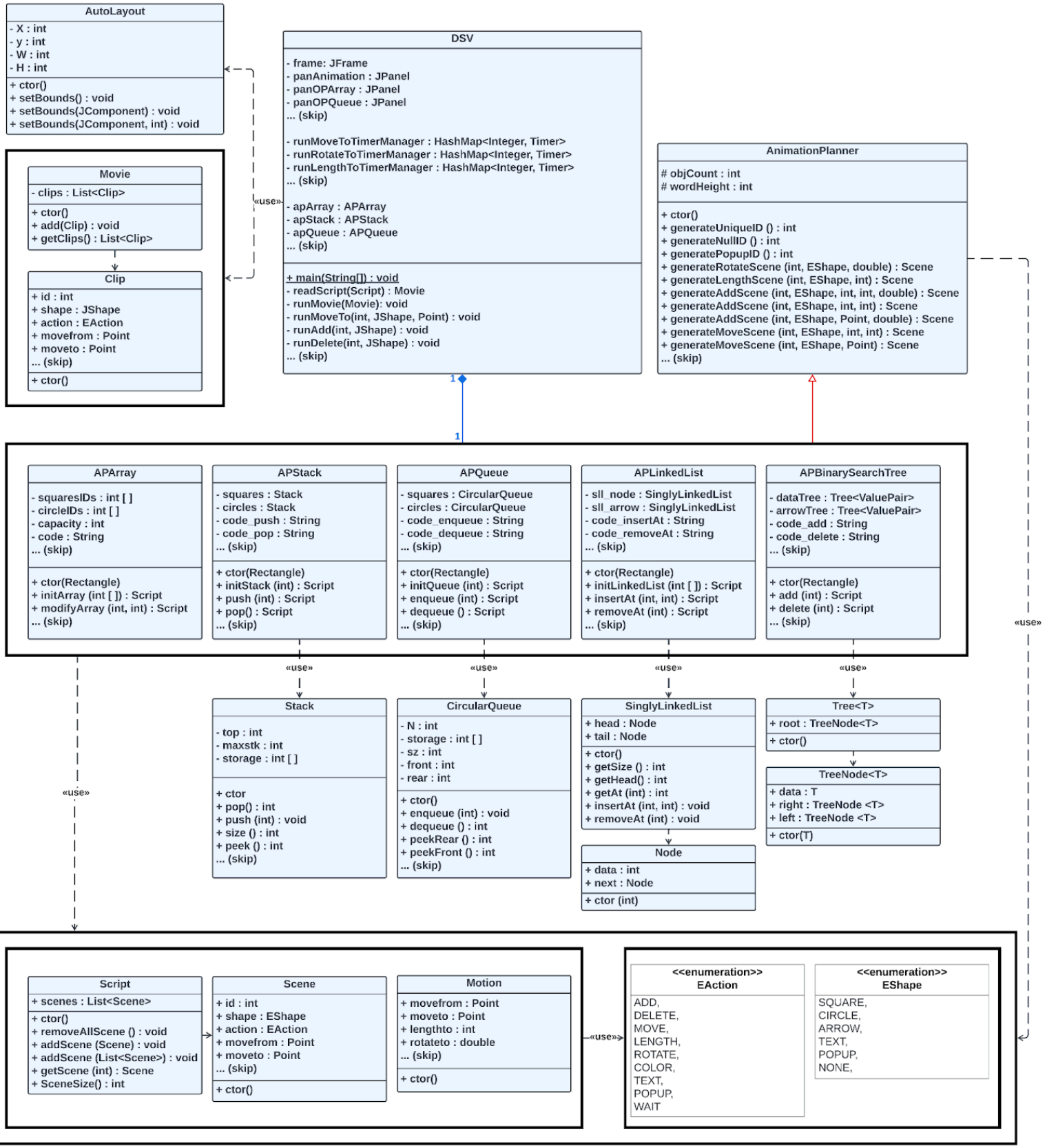
Please note that due to the size of the screen, the height of a tree is constrained to 5.

- Add
    - This function is used to add a node to a binary tree. Please provide a value to the add field and press the “Add” button to perform the add operation.
- Delete
    - This function is used to delete a node from a binary tree. Please enter the number of a node you want to remove. Press the “Delete” button to perform the remove operation.



Implementation

Class Diagram

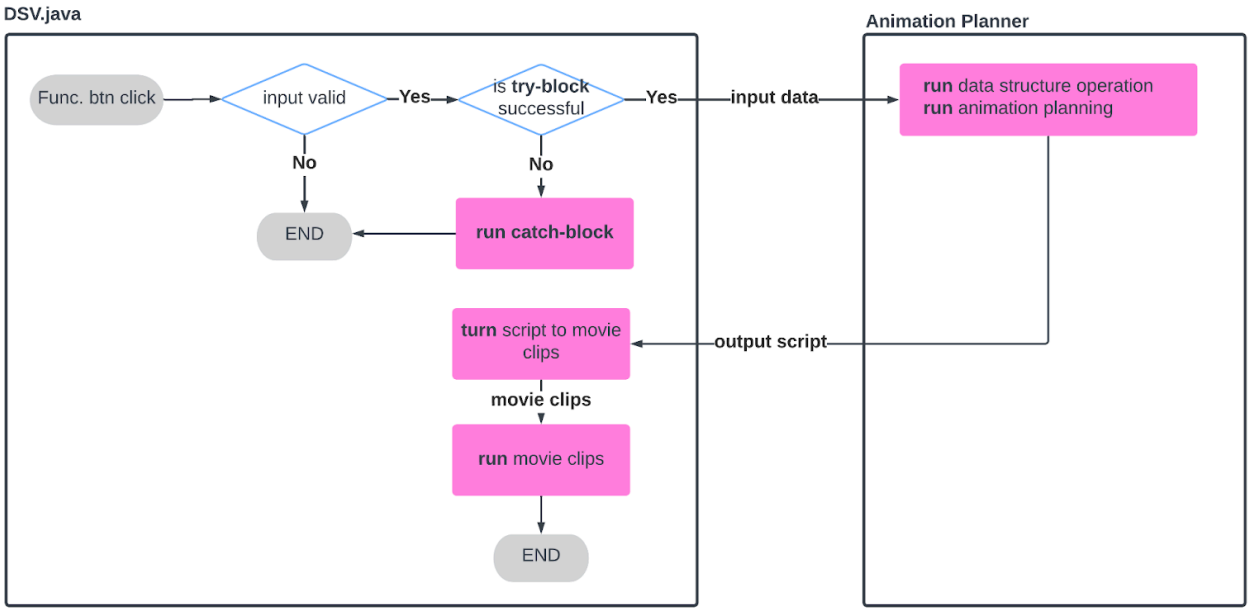


The class diagram gives a birdview of the relationships between objects. Only the important ones will be listed here.

- **DSV**: An object that serves the purpose of receiving and filtering user input, and passing the input to the respective animation planner.
- **Animation Planner**: A base class for all animation planners. In which, it includes all necessary methods to generate scenes (a smaller unit of Script).
- **APArray**: An animation planner for array data structure.
- **APStack**: An animation planner for stack data structure.
- **APQueue**: An animation planner for queue data structure.
- **APLinkedList**: An animation planner for linked list data structure.
- **APBinarySearchTree**: An animation planner for binary search tree data structure.
- **Scene**: A smaller unit of Script. Scene uses a textual way to describe how an object should be moved, such as “node A should move from (0,0) to (10,10)”. Its counterpart is Clip.
- **Script**: A collection of scenes. Its counterpart is Movie.
- **Clip**: A smaller unit of Movie. Clip possesses a real graphical object, such as a circle, a square or other. Clip also records what the object should do.
- **Movie**: A collection of clips. DSV will run each clip of a movie (or run movie clips) one by one to make it feels like the animation is running.

Flow Chart

All of the operations of available data structures in the DSV, such as initializing, adding, deleting and more, can be described by the following flow chart:



Use an array operation for example. When initializing an array, the user input is checked. Then the input string will be parsed inside a try-catch statement. If everything goes right, the input will be passed to APArray as a parameter. Then a script will be generated and further converted to a movie (a collection of movie clips).

Animation

It is time to dig deep for more details about how the animation is implemented. Please recall the whole process:

1. DSV receives user input.
2. DSV passes the input to a respective animation planner, such as APArray.
3. Animation planner performs data structure operations, such as inserting an item, and generating an animation script.
4. Animation planner returns a script.
5. DSV translates a script into a movie.
6. DSV runs a movie.

Step 1 is straightforward, no need for explanation. Step 2 to Step 4 has been discussed in detail in “Graphical Design → Animation”, we will skip them to avoid duplication.

In this section, we will talk about step 5 and step 6. Or specifically, How to turn a script into a movie and How to run a movie.

How to turn a script into a movie?

A script is a textual way of recording how a node should move, each descriptive movement is a scene.

For example, a script may look like this:

```Script
Scene 0 : Add a node.
Scene 1 : Move the node to point (0,0).
Scene 2 : Move the node to point (10, 10).
```

Once a script is returned to DSV, DSV will read that script and transform it into movie clips.

Here's the pseudo code of the readScript method.

```pseudo
Algorithm movie readScript (script){
        New a movie;
        For each scene in a script do:
                New a clip;
                clip.shape = new JShape(); // clip possess a real graphical obj
                clip.movefrom = scene.movefrom;
                clip.moveto = scene.moveto;
                // ... (skip)
                movie add a clip
        Return movie;
}
```

This is how a movie (or movie clips) is made. Each scene is transformed into a clip. Once all the clips are added to a movie, the movie is complete. In general, the only difference between a scene and a clip is that a scene doesn’t possess the real object of a node, whereas a clip does.

How to run a movie?

To avoid blocking the UI thread, timers are employed.
Each clip's movement is controlled by its own timer. For instance, clip 1: "Circle A moves to (0,0)" has its own timer, and so does clip 2: "Circle B moves to (10,10)".

We use the ScheduledExecutorService object to manage the delay execution between each clip, therefore ensuring each clip starts at the appropriate time. By doing so, each clip can be run with desired order and make it feel like an animation is running.

Conclusion

After a month of hard work, along with numerous testings and bug fixes, we are glad that we have completed our Data Structure Visualizer. It is not the best DSV out there in the world, but at least it can withstand the edge condition testing, such as removing the only remaining node from a list, or adding a node to an empty list, and so on. It delivers the functionality as expected. On the screen, a code indicator tells users where the current code is running, and with respective animation executing. Furthermore, it provides a speed control option to users, which allows them to adjust the speed according to their needs. Overall, it is a huge project for us in terms of its scale, but we are glad we have done it on time and it works like a charm.

References

- Prof. Umme Zakia. (2024). Arrays [Slide show]. NYIT, Vancouver, British Columbia, Canada.
- Prof. Umme Zakia. (2024). DS Lec 03 [Slide show]. NYIT, Vancouver, British Columbia, Canada.
- Prof. Umme Zakia. (2024). CSCI 507 Lec 04 [Slide show]. NYIT, Vancouver, British Columbia, Canada.
- Prof. Umme Zakia. (2024). DS Lec 5 [Slide show]. NYIT, Vancouver, British Columbia, Canada.
- Prof. Umme Zakia. (2024). DS Lec 06 Tree [Slide show]. NYIT, Vancouver, British Columbia, Canada.

Appendices

Source Code

- The source code has been uploaded to github. Please visit the following link: https://github.com/Xyberonzyx7/NYIT.CSCI-507-Data-Structures.FinalProject

User Manual

1. Open a command line.
2. Clone the repository using the following command

```
git clone https://github.com/Xyberonzyx7/NYIT.CSCI-507-Data-Structures.FinalProject
```

3. Go to the cloned repository

```
cd NYIT.CSCI-507-Data-Structures.FinalProject
```

4. Compile the DSV source code

```
javac DSV.java
```

5. Run the DSV application

```
java DSV
```

6. The DSV application should be running.