University of Glasgow | School of Computing Science

Algorithm Animator

Andrei-Mihai Nicolae

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 23, 2017

Abstract

Understanding algorithms is both very common and hard for developers in general, regardless of their level of expertise. Even the fundamental ones, such as Dijkstra's algorithm for finding the shortest path between two nodes in a graph, are quite complicated to grasp. Many studies show that visualizing an algorithm and its steps make understanding it much easier. In this report, we will present an Algorithm Animator built specifically for solving this problem in a modern, responsive and efficient manner. Among others, we will also show why certain design decisions (e.g. making it a native desktop app instead of a basic jar, using material design for the user interface), the implementation choices and the evaluation results make this tool a viable option for software engineers when it comes to learning different kinds of algorithms.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Contents

Chapter 1

Introduction

Firstly, the starting point should be defining what an algorithm is.

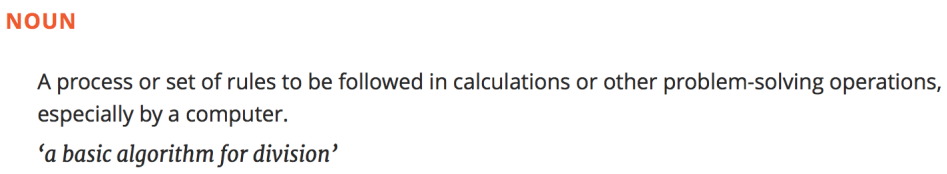

Figure 1.1: Algorithm definition taken from the Oxford English Dictionary.

As we can see above, it is a ”process or set of rules” followed by a computer when trying to solve a problem. As a machine is, at its core, composed of 1s and 0s, a human being needs to visualise what is going under the hood in order to comprehend the steps that are undertaken. In a study conducted by Cristopher D. Hundhausen et al. [5], it was proven that even though algorithm visualisation is not the ”perfect” solution, it is definitely effective in teaching students and experienced developers.

There are many tools out there that already provide this functionality, but one thing was a key factor in deciding how to proceed with planning the whole software process: there are barely any standalone applications. Even though one can find many websites which let people visualise algorithms' steps (e.g. a great example is VisuAlgo [3]), this is not such a good solution as the user cannot use the application at his/her commodity, an Internet connection being required.

Therefore, the plan was to create an app that can be run offline and would feel natural to the user regardless of the operating system of choice. There are many Java implementations (i.e. resulting in jar executables), but one of the main drawbacks is the lack of proper GUI tools that can build native-feeling applications.

As such, the decision was made to use the Electron framework originally built by the team behind the Atom text editor. This framework uses only web tools to generate executables for all 3 main OS families (i.e. macOS, Windows and Linux) that provide the user with a native, modern and responsive feel.

The planning was made using the best agile practices [4], eventually deciding on following a variation of XP programming. Designing and coming up with an architectural plan was a major milestone to be reached as it took a considerable amount of time to be put in place. However, as it will be discussed in future chapters, the implementation of the animation engine was a crucial and time consuming challenge that required the highest amount of effort.

The report also presents how the app was test in many various ways, as well as how it was evaluated using potential end users. In the end, we shall discuss about the roadmap of the project, why open source is and will

be vital for the development of the animator as well as some final thoughts and lessons learned throughout the process.

1.1 Aims

The goals of the project were set and subsequently refined throughout many project meetings, as well as meeting with some fellow classmates to get feedback along the way.

The main aims of the Algorithm Animator, however, have always been:

- Create an efficient and easy-to-use animator.
- Make the animator a cross-platform application that would run natively on the main operating system families: macOS, Windows and Linux.
- Provide a user-friendly interface that would make the user want to enjoy the product.
- Create at least 5-6 fully functional algorithms.
- Make the application scalable and easy to maintain.
- Test all functionality and ensure quality above quantity.
- Evaluate the product throughout the software process to meet acceptance criteria.
- Build a roadmap that would allow the animator grow even after the level 4 project has finished.

1.2 Motivation

Even if we have previously mentioned that visualising algorithms is one of the best ways of understanding how they perform, there are also other key motivational aspects behind the need of building an animator.

One such motivation is the increasing demand of software engineers throughout the industry as well as the rise in level of expertise. Developers, and students in particular, will benefit tremendously from a good grasp on how to implement correct and optimal algorithms when applying for a new job. Thus, an algorithm animator would be a good component in one's tool belt.

Another major aspect was the opportunity to create a modern tool that would make users enjoy working with it. At the moment, one of the most popular animator toolkits is the one presented in John Morris' paper [6]. Below a screenshot was taken from the app.

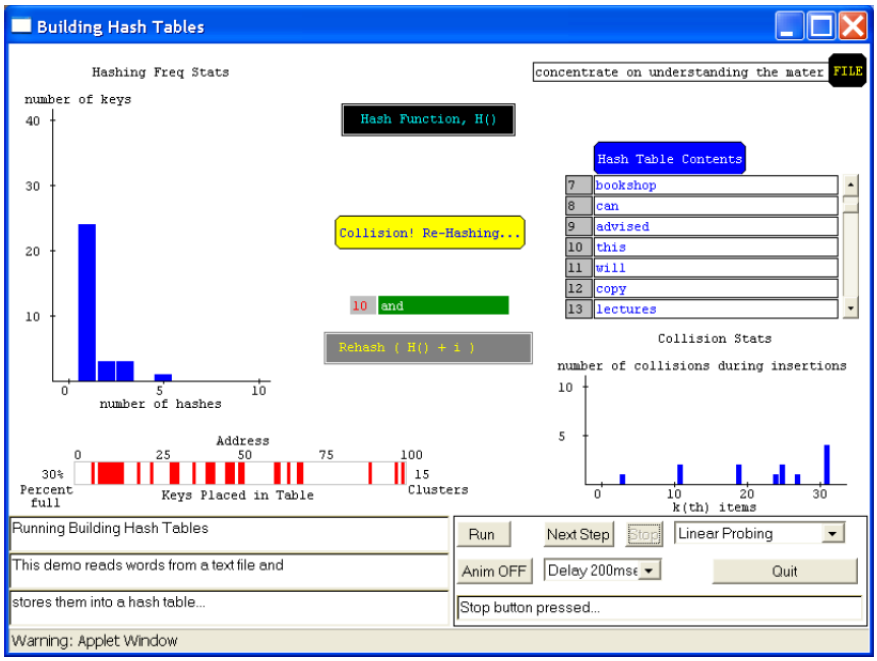

Figure 1.2: John Morris' Animator Toolkit.

It was written in Java, which comes with certain advantages: multi-threading support, enhanced familiarity due to university coursework and projects etc. However, the lack of proper graphical user interface components makes this a not so viable option for regular users when adopting it.

1.3 Contributions

This report serves to deliver certain main contributions:

- Shows the whole software development lifecycle of an algorithm animator.
- Presents various technologies used to build native and modern desktop apps [1] [2]
- Provides a brief overview of current animators available and background knowledge on their benefits and efficiency in teaching.

1.4 Report Content

The rest of the report will analyse the background of animators and why they were proven useful, as well as cover all the steps in gathering requirements, designing, implementing, testing and evaluating the tool.

- Chapter 2 covers work related to the purpose of algorithm animators and why they are useful
- Chapter 3 goes into how the problem was analysed and what requirements were gathered through project meetings and discussions with Algorithmics students.
- Chapter 4 shows the steps undertaken to follow the best agile methodology principles.

- Chapter 5 explains the design decisions behind the tool and illustrates various lessons learned and problems faced along the way.
- Chapter 6 goes into the implementation details of the animator.
- Chapter 7 show how extensive unit, integration and other types of testing (e.g. smoke, end-to-end) were undergone and why they were essential to the development of the application.
- Chapter 8 details the overall results of the project.

Chapter 2

Background

2.1 Related Work

Chapter 3

Requirements

3.1 Problem Analysis

3.2 Requirements Gathering

3.3 Functional Requirements

3.4 Non-Functional Requirements

Chapter 4

Planning

4.1 Agile

4.1.1 Kanban Board

4.1.2 Issues & Bug Tracking

Chapter 5

Design

5.1 Architecture

5.1.1 EDA (Event-driven Architecture)

5.2 Native Desktop App vs. Jar

5.3 Electron

5.4 Vis.js

5.5 Material Design

5.6 Compromises

Chapter 6

Implementation

6.1 Project Structure

6.2 JavaScript and Multi-Threading

6.3 JS Animation Engine

6.4 Extra Features

6.5 Lessons Learned

6.6 Issues Faced

Chapter 7

Testing

7.1 Unit Testing

7.2 Integration Testing

7.3 Prototype Evaluation

7.4 Results

Chapter 8

Conclusions

8.1 Open Source

8.2 Project Roadmap

8.3 Final Thoughts

8.4 Acknowledgements

Bibliography

[1] Electron Website. `https://electron.atom.io`.

[2] VisJs Website. `http://visjs.org`.

[3] VisuAlgo Website. `http://visualgo.net`.

[4] Bharat Choudhary and Shanu K Rakesh. An Approach using Agile Method for Software Development. *Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*, 2016.

[5] Sarah A. Douglas Cristopher D. Hundhausen and John T. Stasko. A Meta-Study of Algorithm Visualization Eectiveness. *Journal of Visual Languages and Computing*, 13:259–290, 2002.

[6] John Morris. A toolkit for algorithm animation. *Proceedings AEESEAP*, 2004.