Colour-Chisel

CPSC 448 Final Report

Student: Christopher Powroznik (44281146)

Faculty Supervisor: Elisa Baniassad

Table of Contents

[Overview 2](#_Toc20217387)

[Learning Objectives 2](#_Toc20217388)

[Survey of Prior Work 2](#_Toc20217389)

[The Existing Colour Patterns 2](#_Toc20217390)

[Compliments 2](#_Toc20217391)

[Analogous 2](#_Toc20217392)

[Triadic 2](#_Toc20217393)

[Tetradic 2](#_Toc20217394)

[Primary Secondary and Tertiary Schemes 2](#_Toc20217395)

[The ‘S’ Pattern 3](#_Toc20217396)

[Shape Based Selection 3](#_Toc20217397)

[What is Colour Chisel? 3](#_Toc20217398)

[Milestones 3](#_Toc20217399)

[Create an EBNF 3](#_Toc20217400)

[Summary 3](#_Toc20217401)

[Reflection 4](#_Toc20217402)

[Bibliography 4](#_Toc20217403)

# Overview

The following is a document outlying the learning objectives, development and overall progression Christopher Powroznik CPSC 448 project ‘Colour Chisel’.

# Learning Objectives

As agreed between the Elisa Baniassad and Christopher Powroznik. The following learning objectives are the goals for the project. By the end of the project Christopher will have covered all of the following…

* Learn how to develop a continuous integration setup involving automated testing, publishing and deployment of a web demo as a hook from committing code.
* Learn to integrate a compilation tool into development pipelines such as web pack and bower
* Learn geometric transformations in a circular space programmatically to do operations on the colour wheel
* Learn to setup and develop a project without a boilerplate
* Learn to create procedural usage libraries that recursively repeats functionality
* Explore different colouring techniques programmatically as opposed to traditional primary, secondary and tertiary design
* Deepen my learning to iteratively make code design choices and refactor code based on feedback and direction of the project.
* Deepen my learning of developing test suites to perform unit testing, end-to-end testing and black box testing of my code

# Survey of Prior Work

## The Existing Colour Patterns

It is quick to find that there are defined ways to finding colours that look good together, which are even mentioned in Isaac Newton’s initial paper on the colour spectrum (Newton, 1665). All of these have inspired the default actions in my Colour Chisel library.

### Compliments

Compliment colours are colours that are directly across from each other on the colour wheel. They contrast entirely with and appear more prevalent when next to each other.

### Analogous

An analogous colour is one that is a rotation from one colour to another that is usually just to the left or right of the first colour. These are commonly used as accents, in order to highlight slight changes and create a deeper colour, by including a left or right analogous of a primary colour.

### Triadic

### Tetradic

## Primary Secondary and Tertiary Schemes

## The ‘S’ Pattern

# Shape Based Selection

# What is Colour Chisel?

# Milestones

## Create an EBNF

(September 19, 2019 🡪 )

Define and construct an EBNF for the colour chisel language. This will be the blueprint to work off of for the implementation process. This EBNF is a living file and can be edited as the project continues and scope changes.

### Summary

I started out by trying to find the most standard way to produce an EBNF. I came across the article ‘the language of languages’ (Might, n.d.). It covered writing an EBNF in a very robust methodology. I familiarized myself with the syntax of an EBNF by following examples from Wikipedia (Extended Backus–Naur form , n.d.). I began writing my EBNF and checking my work using ‘Railroad Diagram Generator’ (Rademacher, 2019). I first created a base EBNF, that was very barebones, especially with pathing. I wasn’t sure on how to handle variables and getting certain colours yet.

expression  
 ::= (const | path) ("," manipulation)\*  
  
/\* Pathing \*/  
path ::= "[" (expression) ("," expression)\* "]"  
  
/\* Manipulations \*/  
manipulation  
 ::= analogous  
 | compliment  
analogous  
 ::= (A|a) range?  
compliment  
 ::= C  
 | c  
range ::= '-'? [0-9]+ ("." [0-9]+)?  
  
/\* Constants \*/  
const ::= '(' ( rgb | rgba ) ')'  
rgba ::= 'rgba(' colourValue ',' colourValue ',' colourValue ',' colourValue ')'  
rgb ::= 'rgb(' colourValue ',' colourValue ',' colourValue ')'  
hex ::= '#'? hexit hexit hexit ( hexit hexit )?  
hexit ::= [0-9a-fA-F]  
colourValue  
 ::= ( [1-9] | '1' [0-9] | '2' [0-4] )? [0-9]  
 | '25' [0-5]  
rgba ::= 'rgba(' colourValue ',' colourValue ',' colourValue ',' colourValue ')'

After reflecting and reading the article \_\_\_\_\_. I decided to edit my original definition of expression to handle the case of a const or path being a saved variable. This led me to the new EBNF shown below. I handled paths in a similar way to JavaScript arrays. Where defining a path, I could reference a spot in the path by calling the path or variable and indicating the index of the colour I wanted to take from the path.

expression  
 ::= (const | path | variableName) ("," manipulation)\*  
  
/\* Variables \*/  
variable ::= var variableName "=" (expression | path)  
pathVariable ::= variableName "[" [0-9]+ "]"  
variableName ::= [a-fA-F]+  
  
/\* Manipulations \*/  
manipulation  
 ::= analogous  
 | compliment  
analogous  
 ::= (A|a) range?  
compliment  
 ::= C  
 | c  
range ::= '-'? [0-9]+ ("." [0-9]+)?  
  
/\* Constants \*/  
const ::= '(' ( rgb | rgba ) ')'  
rgba ::= 'rgba(' colourValue ',' colourValue ',' colourValue ',' colourValue ')'  
rgb ::= 'rgb(' colourValue ',' colourValue ',' colourValue ')'  
hex ::= '#'? hexit hexit hexit ( hexit hexit )?  
hexit ::= [0-9a-fA-F]  
colourValue  
 ::= ( [1-9] | '1' [0-9] | '2' [0-4] )? [0-9]  
 | '25' [0-5]  
rgba ::= 'rgba(' colourValue ',' colourValue ',' colourValue ',' colourValue ')'  
  
/\* Pathing \*/  
path ::= "[" (expression) ("," expression)\* "]"  
rotate ::= expression ".rotate(" range ")"  
translate  
 ::= expression ".translate(" colourValue "," colourValue ")"  
scale ::= expression ".scale(" colourValue ")"

### Reflection

# Bibliography

*Extended Backus–Naur form* . (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Extended\_Backus%E2%80%93Naur\_form

Might, M. (n.d.). *the language of languages*. Retrieved from mattmight.net: http://matt.might.net

Newton, I. (1665). Newton’s Color Theory.

Rademacher, G. (2019, Augst 19). *Railroad Diagram Generator*. Retrieved from bottlecaps: https://bottlecaps.de/rr/ui