Inhaltsverzeichnis

1. Einführung und Ziele

Dies ist meine Doku über meinen Image / Color Transpiler

1.1. Aufgabenstellung

1.1.1. Was ist der Image Transpiler?

- Liest ein Image ein und wandelt es in Brainfuck um
- Je nach dem welche Farbe der aktuelle Pixel im Bild hat, wird der Brainfuck Code anders aussehen

1.1.2. Wesentliche Features

- Jeder Pixel eines Bildes wird auf Farben gecheckt
- Farben sind verschiedene Brainfuck Zeichen
- Brainfuck Code wird ausgegeben
- Brainfuck Code wird interpreted und das Resultat wird ausgegeben

1.1.3. Qualitätsziele

Qualitätsziele	**Erklärung**
Image Transpiler	*Einen erfolgreichen Image zu Brainfuck transpiler herstellen*
Brainfuck Validierung	*Eine Methode erstellen, welche den Brainfuck-Code validiert*
Brainfuck Interpreter	*Einen Interpreter schaffen, welcher von einem Brainfuck Code ein Resultat ausgibt.*

1.2. Stakeholder

Der Auftrag war es, selber einen Interpreter, Transpiler oder einen Compiler mit einer beliebigen Sprache zu programmieren. Man konnte auch entscheiden, ob man eine Sprache komplett neu erfinden will oder einfach eine Sprache zu einer anderen zu übersetzen. Ich habe mich dazu entschieden, einen Transpiler und einen Interpreter zu schreiben.

Rolle	**Kontakt**	**Erwartungshaltung**
Lehrer	*Herr von Känel*	*Sollte ein Interpreter für eine Esolang Sprache / eigene Sprache sein.*

Rolle	Kontakt	Erwartungshaltung
Entwickler	-	*Können ein Bild im Code angeben und dann wird jeder Pixel gecheckt, ob er eine passende Farbe hat. Wenn das so ist wird der Pixel zu einem Brainfuck Zeichen umgewandelt und am Schluss wird der ganze Brainfuck Code ausgegeben. Danach wird der Brainfuck Code durch einen Interpreter ausgebeben.*
Autor	*Timo Angst*	_Eine gute Note mit meinem Projekt erreichen zu können.

2. Randbedingungen

Die Entscheidung, wie oder über was wir unser Projekt machen, war uns frei. Zu beachten gab es nur, dass man entweder einen Compiler, einen Transpiler oder einen Interpreter programmieren musste. Für das Projekt hatten wir circa 4 Wochen Zeit, also in etwa 12 Lektionen (abzüglich Erklärungen etc). Die Zeit war extra so auf die 4 Wochen aufgeteilt, damit die Lehrer unser Projekt noch vor den Sommerferien korrigieren können. Somit fliesst diese Note noch in das jetzige Zeugnis ein.

Randbedingungen	**Erläuterung**
Thema	Das Thema war einem frei, ausser dass es entweder ein Interpreter, Transpiler oder Compiler sein muss. Ich habe eine Mischung davon gemacht - einen Transpiler und dann noch einen Interpreter.
Zeitlimit	4 Wochen
Team	Einzelarbeit
Abgabe	Die Doku muss als PDF abgegeben werden. Der Rest ist auf Github
Testen	Keine Tests

3. Kontextabgrenzung

3.1. Fachlicher Kontext

3.1.1. User

- Lädt eine Bild Datei in das Programm und startet es

3.1.2. Transpiler

- Der Transpiler checkt jeden Pixel des angegebenen Bildes auf eine der folgenden RGB's
  - 255, 255, 255
  - 0, 0, 0
  - 0, 255, 0
  - 255, 0, 0
  - 0, 0, 255
  - 128, 128, 128
  - 255, 255, 0
  - 0, 255, 255

Je nach dem, welche RGB der momentane Pixel hat, wird ein anderes Brainfuck Zeichen ausgegeben.

3.2. Compiler

- Sobald der Brainfuck Code vom Transpiler angekommen ist, kontrolliert ihn der Compiler auf Syntax-Errors oder andere Fehler im Code. Bei korrektem Code wird er compiled und das Resultat wird ausgegeben.

3.3. Ausgabe

- Über den ganzen Prozess hinweg wird einmal der Brainfuck Code, allfällige Fehler und das Resultat ausgegeben

3.4. Technischer Kontext

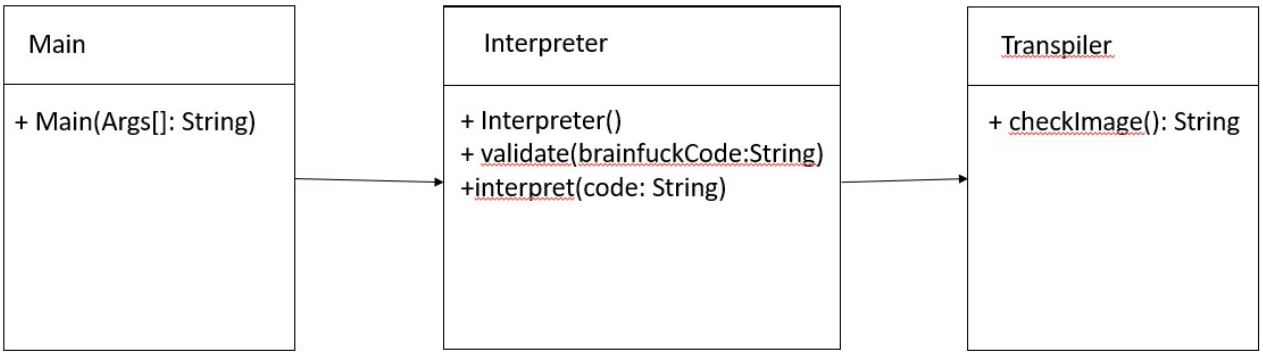
- Der Code ist mit Java geschrieben und verwendet ein .png oder .jpg Image

4. Lösungsstrategie

Zuerst habe ich im Internet ein bisschen recherchiert, um herauszufinden, wie man ein Bild Pixel für Pixel einlesen kann und die Farbe bekommen kann, dann habe ich angefangen am Transpiler zu arbeiten. Sobald ich diesen lauffähig hatte, habe ich eine Compiler Klasse erstellt, welche den Brainfuck Code vom Transpiler verarbeitet.

5. Bausteinsicht

5.1. Whitebox Gesamtsystem



5.1.1. Main()

- Erstellt die Klasse Interpreter

5.1.2. Interpreter()

- Konstruktor von der Interpreter Klasse → wird aufgerufen sobald die Klasse erstellt wird

5.1.3. validate()

- Ruft die Methode checkImage() vom Transpiler auf und validiert den erhaltenen Brainfuck Code

5.1.4. interpret()

- Interpreted den Brainfuck Code bei erfolgreicher Validierung und gibt das Ergebnis aus

5.1.5. checkImage()

- Scannt Pixel für Pixel aus dem Bild und erstellt daraus den Brainfuck Code

6. Laufzeitsicht

6.1. *Erfolgreich*

- 1. Bild angeben
- 2. Programm starten
- 3. Bild wird gescanned - jede gültige Farbe wird zum Brainfuck Code appended
- 4. Brainfuck Code wird validiert
- 5. Brainfuck Code okay - Code wird interpretiert
- 6. Ergebnis wird ausgegeben

6.2. *Bild nicht valid*

- 1. Bild angeben
- 2. Programm starten
- 3. Bild ist nicht valid
- 4. Programm gibt einen Error aus

6.3. *Brainfuck-Code invalid*

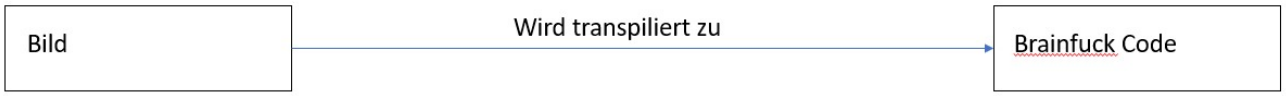
- 1. Bild angeben
- 2. Programm starten
- 3. Bild wird gescanned - jede gültige Farbe wird zum Brainfuck Code appended
- 4. Brainfuck Code wird validiert
- 5. Brainfuck Code nicht okay
- 6. Meldung wird ausgegeben

6.4. *Brainfuck-Code ohne Ausgabe*

- 1. Bild angeben
- 2. Programm starten
- 3. Bild wird gescanned - jede gültige Farbe wird zum Brainfuck Code appended
- 4. Brainfuck Code wird validiert
- 5. Brainfuck Code okay - aber gibt keine Ausgabe
- 6. Nichts wird ausgegeben

7. Verteilungssicht

7.1. Infrastruktur Ebene 1



Begründung

Der Transpiler transpiliert das eingebette Bild zu Brainfuck Code

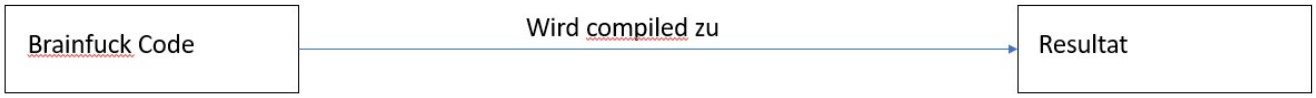
Qualitäts- und/oder Leistungsmerkmale

Es wird schnell und unkompliziert transpiliert

Zuordnung von Bausteinen zu Infrastruktur

Es wird beim Start des Programms gemacht

7.2. Infrastruktur Ebene 2



Begründung

Der Compiler compiled den Brainfuck Code zu einem Resultat

Qualitäts- und/oder Leistungsmerkmale

Es wird schnell und unkompliziert kompiliert

Zuordnung von Bausteinen zu Infrastruktur

Es wird zum Schluss des Programms gemacht

8. Querschnittliche Konzepte

8.1. *Versionskontrolle*

Die Versionskontrolle von diesem Projekt ist GitHub

8.2. *Fehlerbehandlung*

Fehler werden durch verschiedene Wege behandelt. Hier eine kleine Tabelle dazu.

Fehler	Folge
Ungültige Farbe	*Pixel wird ignoriert*
Ungültiger Brainfuckcode	*Eine Meldung wird ausgegeben: "Brainfuckcode ist ungültig"*
Brainfuckcode ist valid, aber Resultat leer	*Keine Ausgabe des Resultats erfolgt*

9. Architekturentscheidungen

9.1. Console Output

Mein Programm kann gesteuert werden über die Konsole. Dies habe ich so gemacht, da wir nicht viel Zeit hatten und es so auch praktisch ist, da nichts extra starten muss.

Eine andere Möglichkeit wäre ein GUI gewesen, welches für einen User viel einfacher ist um anzuwenden. Jedoch wird dieses Programm sowieso nicht von einem normalen User genutzt werden.

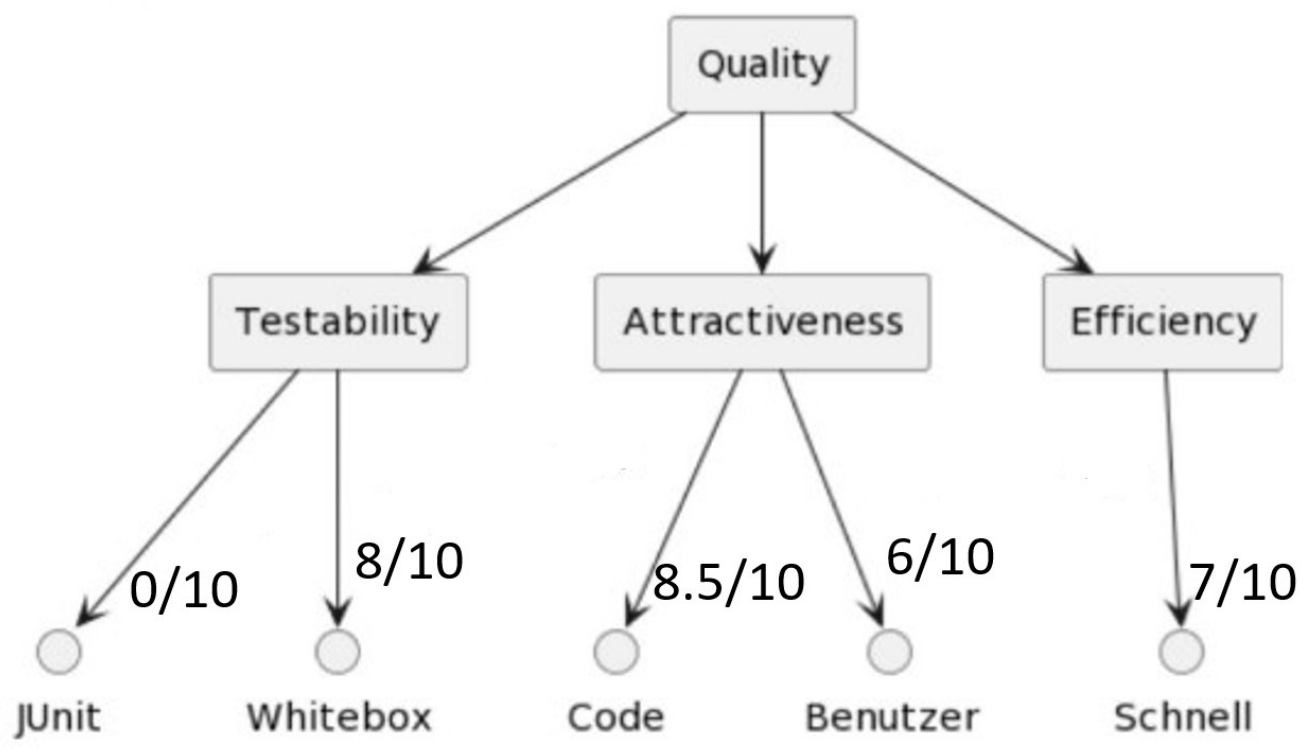
Konsole	**GUI**
+ Simpel	+ User Friendly
+ Weniger Zeitaufwand	+ Schöner
- Nicht wirklich User Friendly	- Viel Zeitaufwand

9.2. Interpreter / Transpiler / Compiler

Ich habe mich entschieden, einen Transpiler und einen Compiler zu machen, da ich ein bisschen mehr machen wollte, als nur einen langweiligen Compiler / Transpiler. Dazu habe ich eine Idee umgesetzt, welche ich so noch nirgens gesehen oder gehört habe. Davor wusste ich gar nicht, dass man durch ein Bild etwas Transpilen kann.

10. Qualitätsanforderungen

10.1. Qualitätsbaum



10.1.1. Bewertungen

- JUnit 0/10 → Ich habe keine JUnit Tests geschrieben, da wir dafür zu wenig Zeit bekamen
- Whitebox 8/10 → Ich habe selber beim schreiben vom Code sehr viel getestet
- Code 8.5/10 → Mein Code ist übersichtlich, objektorientiert und strukturiert
- Benutzer 6/10 → Es gibt kein GUI aber ist trotzdem simpel zu bedienen
- Efficiency 7/10 → Das Programm ist sehr schnell, solange man nicht 4k Bilder oder so mitgibt

10.2. Qualitätsszenarien

Szenario	Folge
Ungültige Farbe	*Pixel wird ignoriert*
Ungültiges Bildformat	*Error wird geworfen*
Ungültiger Brainfuckcode	*Eine Meldung wird ausgegeben: "Brainfuckcode ist ungültig"*
Brainfuckcode ist valid, aber Resultat leer	*Keine Ausgabe des Resultats erfolgt*

11. Risiken und technische Schulden

- Falls man ein sehr grosses Bild hat (mit ca.4k Pixeln oder so) wird die Wartezeit vermutlich sehr lange sein, da der Code jeden einzelnen Pixel kontrollieren muss.
- Bei einer ungültigen Bilddatei könnte das Programm crashen oder einen Error ausgeben

12. Glossar

Begriff	Definition
Brainfuck	*Brainfuck ist eine esoterische Programmiersprache, die der Aminet-Gründer, der Schweizer Urban Müller, im Jahre 1993 entwarf.*
Transpiler	*Übersetzt eine Programmiersprache in eine andere*
Compiler	*Computerprogramm, das von einer Programmiersprache in eine andere übersetzen kann*
Interpreter	*Als Interpreter wird ein Computerprogramm bezeichnet, das eine Abfolge von Anweisungen anscheinend direkt ausführt, wobei das Format der Anweisungen vorgegeben ist.*