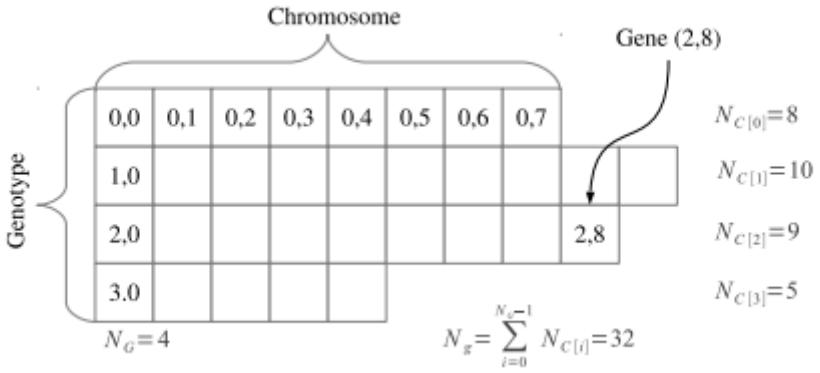
Technologierecherche – Jenetics

„Jenetics is a Genetic Algorithm, Evolutionary Algorithm, Genetic Programming, and Multi-objective Optimization library, written in modern-day Java.“ [1]

Funktionsweise:

Zur Darstellung der Individuen einer Population benutzt Jenetics eine Struktur mehrerer Klassen. Die Klasse *Phenotype* repräsentiert ein einziges Individuum und besteht selbst aus einem *Genotype*, der Generation, in welcher das Individuum erstellt wurde, sowie einem optionalen Fitness-Wert. Der *Genotype* kann aus mehreren *Chromosomes* bestehen, welche wiederum aus mindestens einem *Gene* gebildet werden. Das *Gene* bildet die unterste Ebene eines Individuums und enthält die tatsächliche Information dessen. Eine Population wird mittels des *ISeq* Interfaces umgesetzt, welches eine geordnete Folge bestimmter Größe mehrerer Elemente umsetzt, sie ist also lediglich eine Sammlung mehrerer *Phenotypes* und keine eigener Klassentyp. Jenetics kommt mit Bit, Character, Integer, Long, Double und Enum Gen-Arten, es können aber auch eigene Gen-Arten implementiert werden. Die Modellierung eines zum Problem passenden *Genotypes* ist wahrscheinlich der wichtigste Schritt bei der Entwicklung eines evolutionären Algorithmus mit Jenetics. [2]



Struktur des *Genotype* [2]

Die Evolution einer Population wird von einer *Engine* umgesetzt, welcher verschiedenste Parameter für die Umsetzung dieser zugewiesen werden können und erfolgt in einem *EvolutionStream*, der von der *Engine* erstellt wird und schlussendlich die entwickelte Population zurückgibt, es kann aber auch direkt von der *Engine* ein einzelner Evolutionsschritt ausgeführt werden. Die Fitnessfunktion, die von der Engine benutzt wird, um die *Chromosomes* zu evaluieren kann/muss selbst definiert werden. [2]

Anforderungen:

Abruf von Statistiken

Da Jenetics *Engine* stets auf einer Population arbeitet und nach jedem Evolutionsschritt die weiterentwickelte Population zurückgegeben werden kann, ist zu jeder Zeit auf alle vorhandenen Individuen zuzugreifen. So können interessante Informationen, wie Fitnesswerte, sowie Alter der *Phenotypes* aber auch die *Chromosomes* bzw. *Genes* selbst eingesehen und evaluiert werden. [2]

Desweiteren verfügt Jenetics über die *EvolutionStatistics* Klasse, welche, auf einen *EvolutionStream* angewandt, benutzt werden kann um an zusätzliche Daten, wie statistische Informationen zur Fitness, fehlerhafte und getötete *Phenotypes*, sowie Laufzeitinformationen zu den unterschiedlichen Evolutionsschritten zu kommen. [2]

Es sollten also genügend Möglichkeiten bestehen, um an Interessante Statistiken zu den einzelnen Evolutionsschritten zu kommen. Zur Visualisierung dieser bietet Jenetics allerdings keine direkte Unterstützung, wir müssen uns also selbst darum kümmern die Daten entsprechend darzustellen.

Pipes and Filters

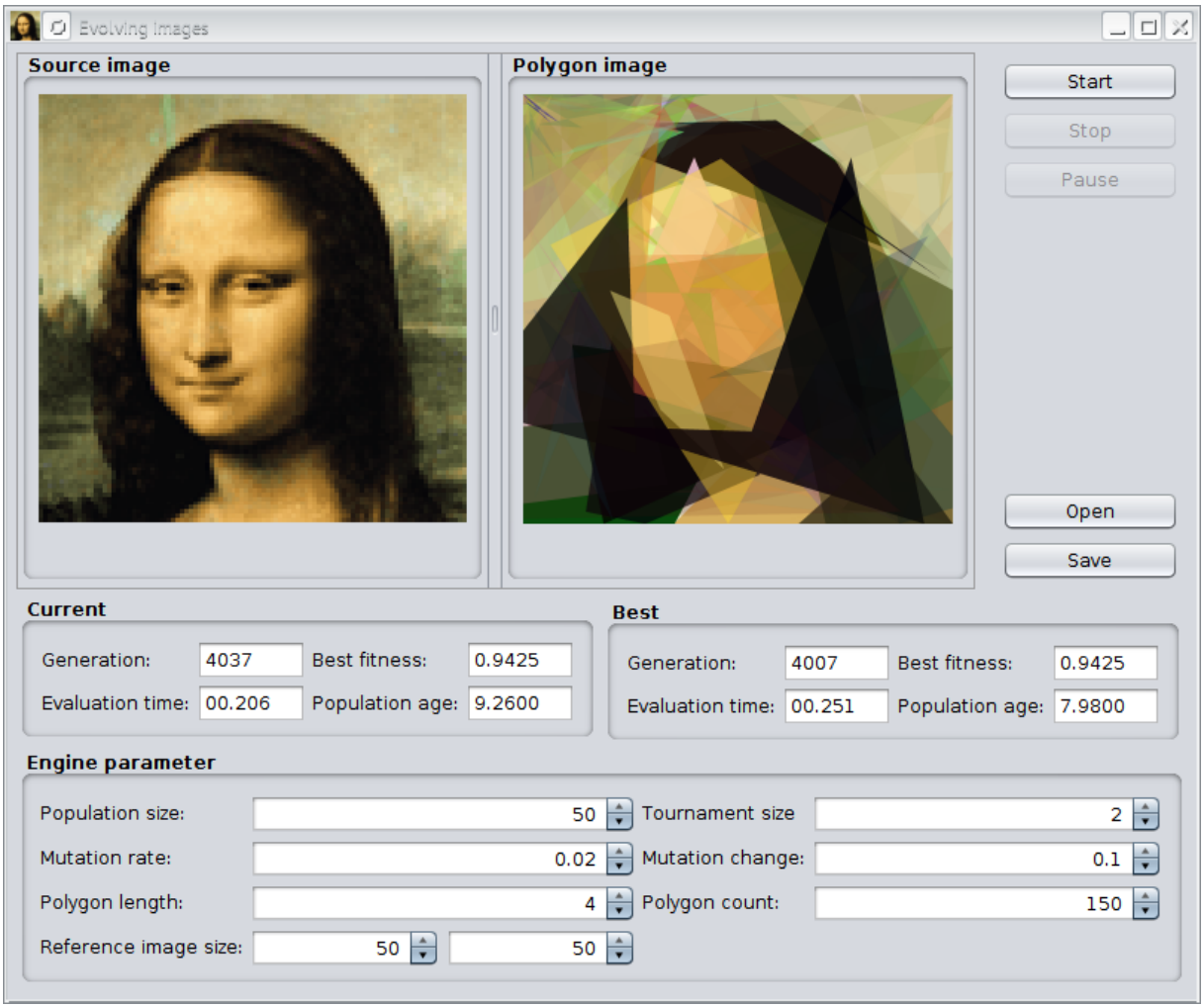
Jenetics bietet dank der Umsetzung des Evolutionsprozesses mit mehreren Klassen an vielen Stellen Möglichkeiten zur Individualisierung dieses, somit ist ein Pipes and Filters Konzept durchaus denkbar.

Die *Engine* ist wohl die Klasse mit den meisten Konfigurationsmöglichkeiten, aber auch am *EvolutionStream* können Einstellungen vorgenommen werden. Des Weiteren können die verschiedenen Gen-Arten ausgeschöpft werden um das Problem auf unterschiedliche Weise zu encoden. [2]

Ein Problem, welches die *Engine* mit sich bringt, ist, dass sie, nachdem sie einmal erstellt wurde, nicht angepasst werden kann. Es kann also beispielsweise nicht während einer laufenden Evolution die Fitnessfunktion angepasst werden. Hierfür müsste man die zuletzt erstellte Population nehmen und einer neuen *Engine* übergeben. [2]

//Bei diesem Punkt bin ich mir nicht zu 100% sicher!!! Es kann sein das Parameter, wie das Selektionsverfahren auch im Nachhinein noch angepasst werden können. Bezogen habe ich mich bei diesem Absatz auf den Satz: *„Once the Engine is created, via ist Builder class, it can't be changed."*, zu finden auf Seite 23 im Handbuch, allerdings weiß ich nicht welche Parameter genau davon betroffen sind.

Beispielprogramm



GUI eines Evolving-Image Programms mit Jenetics [1]

Hier zu sehen ist ein Programm zur Approximation eines gegebenen Bildes mit Polygonen, für das Jenetics verwendet wurde. Ähnlich wie es bei uns später umgesetzt werden soll, kann man hier die Parameter der Evolution verändern. Dies zeigt, dass unser Projekt mit Jenetics umsetzbar sein sollte. [3]

Konfigurationsmöglichkeiten:

Genotype

- BitGene
- CharacterGene
- IntegerGene
- LongGene
- DoubleGene
- EnumGene

Engine (siehe Handbuch, S. 24 – 25 für Erläuterungen)

- Alterer
    1. Mutation (siehe Handbuch, S. 17 für Spezifikationen)
    2. Recombination (siehe Handbuch, S. 18 - 22 für Spezifikationen)
- Clock
- Constraint
- Executor
- Fitness function (siehe Handbuch, S. 22 – 23 für Spezifikationen)
- Interceptor
- maximalPhenotypeAge
- offspringFraction
- Optimize
- populationSize
- Selector (siehe Handbuch, S. 12 – 16 für Spezifikationen)

EngineStream

- Predicates (siehe Handbuch, S. 26 für Spezifikationen)

Fazit:

Jenetics sollte alle nötigen Funktionen liefern, um unser Projekt umzusetzen. Allerdings ist es auch eine recht umfangreiche Bibliothek und die Zeit, die wir uns durch die Benutzung sparen muss, zumindest zu einem Teil, in das Studium des Handbuchs gesteckt werden.

Referenzen:

Dokumentation

Handbuch

Quellen:

[1]: https://jenetics.io/

[2]: https://jenetics.io/manual/manual-6.2.0.pdf

[3]: https://github.com/jenetics/jenetics#evolving-images