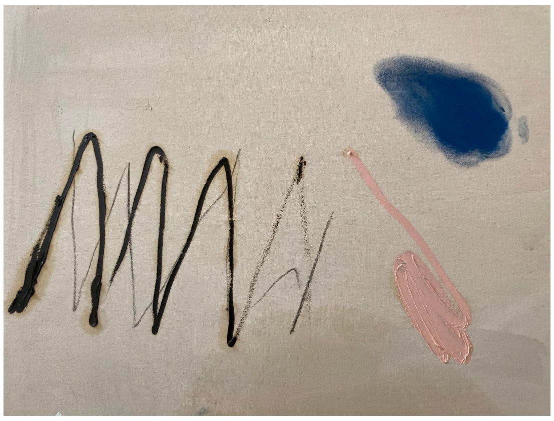
ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

International Hellenic University
Department of Information and Electronic Engineering

MASTER THESIS

Evolduo: A platform for collaborative musical synthesis using evolutionary algorithms



Student
Konstantinos Georgiadis
Reg. Number: 2/2019

Supervisor
Panagiotis Adamidis
Professor

June, 2023

Thesis title: “Evolduo: A platform for collaborative musical synthesis using evolutionary algorithms”
Thesis id: 21284
Student name: Konstantinos Georgiadis
Supervisor name: Panagiotis Adamidis
Thesis assignment date: 07-08-2021
Thesis completion date 10-06-2023

«To my wife Kiki, for her patience and support»

Preface

My motivation for doing this work is multi-faceted and stems from a combination of factors. The opportunity to return to an area where I previously worked 15 years ago is exciting, as it allows me to revisit my past experiences and explore how the field has developed since then. Additionally, the chance to collaborate with a professor with whom I have a great working relationship is highly motivating, as I value his expertise and perspective. Moreover, the focus of the research on art is a source of inspiration for me, as I am deeply interested in exploring the ways this can influence and impact creative expression. By combining these factors, I am driven to pursue this research with dedication and enthusiasm, as I believe it has the potential to make a meaningful contribution to the field of generative art.

Abstract

This postgraduate thesis presents a less common approach to artifical intelligence driven music synthesis. A web application is created that allows multiple users to collaboratively generate music through an evolutionary process. The system uses an evolutionary algorithm to evolve populations of melodies that attempt to match a predefined chord progression. Users can provide feedback on the musical output, which alter system's evaluation. More than 30 people used this platform and provided ratings. The findings of this study indicate that while it is possible to create musically pleasing melodies without extensive knowledge of complex algorithms, developing a user friendly interface and process is a more complicated task. The results suggest that the platform has the potential to facilitate collaborative music generation and could be used in various contexts, such as composition and live performances. Consequently, this study provides a basis for a potential user friendly and interactive web application for musical synthesis.

Acknowledgements

I would like to express my heartfelt gratitude to my wife, Kiki, for her unwavering support and understanding throughout this project. Her love and encouragement kept me going, and I couldn't have done this without her.
I would like to thank Professor Panagiotis Adamidis for his invaluable support and guidance. His exprerience was intrumental in shaping this work to something meaningful and constrained enough to be impelemented.
I am indebted to UI expert Konstantinos Kaloutas for his valuable contributions to the project. His feedback and help on the UI were critical in creating an intuitive and user-friendly experience.
I am also grateful to musician and painter Angelos Krallis for providing feedback on the music generated by this project and for creating the cover drawing for this thesis. His insights and suggestions helped to enhance the musical output and the evolution of this project.
I would also like to thank my friend Stelios Mikedis for his extensive feedback and for the moments we were sharing our academic work updates.
Finally, I would like to thank all the people who used the project and provided feedback. Their input was immensely valuable, and it helped me to understand how the project could be improved in the future.

Contents

List of Figures

List of Code

List of Tables

Chapter 1: Introduction

Musical composition is a complex and creative process that involves the creation of new music using a variety of techniques and tools. In recent years, evolutionary algorithms (EAs) have emerged as a powerful tool for musical composition, and they have been used to create a wide range of music in different styles and genres.

This thesis aims to investigate the potential of evolutionary algorithms in producing music that possesses two distinct characteristics - being pleasant to the ear and easy to remember. The study will delve into the development of methods that can effectively incorporate these traits into the music generated by evolutionary algorithms. Furthermore, the evaluation of these methods will also be conducted to determine their effectiveness in achieving the desired outcomes.

The use of evolutionary algorithms for musical composition offers many advantages over traditional methods, including the ability to search large spaces of musical possibilities, the ability to incorporate constraints and preferences into the music generation process, and the ability to generate a diverse range of musical ideas.

This thesis builds upon previous work done in this particular field, drawing on the author's experience in developing web applications. It aims to provide detailed information of how to design and implement a web-based application utilizing Clojure to generate music using evolutionary algorithms.

This work is divided into four parts to provide a systematic and thorough analysis.

Section 2 focuses on the development of the web application, encompassing an exploration of Clojure fundamentals and the benefits of using a functional language for this work. Additionally, it documents the architecture of the web application, infrastructure details, libraries used and license details.

In Section 3 an introductory exploration of Evolutionary algorithms is presented, including the author's prior contributions in this domain, the libraries used for the implementation, fundamental principles of music theory, the structure of chromosomes, and detailed considerations of fitness function. This section further provides the relationships and connections between these elements in the process of creating algorithmically generated music.

Section 4 presents usage results of the application and discusses the feedback received from users.

Section 5 discusses the selection of tools and methods employed in the project, assessing their effectiveness. It also documents the various challenges encountered and explores potential ways to move forward.

As a final note we should keep in mind that the main motivation behind this work is to accomplish something memorable. Just as Joe Pass explains in his instructional video Jazz Lines[1] "Music that we can't remember isn't worth playing". The goal is to produce compositions that not only possess the potential to be enjoyed by listeners but also have the ability to be remembered long after they have been experienced.

Chapter 2: Web Development

Evolduo is a web application written in the Clojure programming language. It consists of two logical blocks: web and music. The music block comprises two components, music harmony and evolutionary algorithms, which are integrated into the application. If this project reaches maturity, the music component is extracted into a standalone library, enabling other users to use it.

This chapter covers the fundamentals of the Clojure language, the application architecture, a brief explanation of the tools employed, and discuss the process of deploying the application to a virtual private server (VPS).

2.1 Clojure

This section provides some background information on the Clojure programming language, followed by a primer (2.1.2) section which should help the reader understand the code examples presented in the rest of this work. This is by no means a full language guide and may be inadequate for those who have no previous exposure to code from the Lisp family of languages (e.g. Scheme) or other functional languages (e.g. Ocaml). Readers who are familiar with Clojure can skip this section.

2.1.1 Language characteristics

Clojure is a functional programming language designed by Rich Hickey in 2005 and released in 2007. It is designed to be hosted, making it possible to run Clojure code across different runtime environments. Initially, Clojure targeted the Java Virtual Machine (JVM) and the Common Language Runtime (CLR), but the JVM quickly became the primary supported target, while the CLR was supported by the community. Clojurescript is the Clojure implementation that runs on the Javascript runtime. It is the most popular runtime after the JVM and can target browsers and node.js runtimes. Recently, a Clojure dialect has also targeted the Dart runtime [1].

Clojure is dynamically typed, which means that types don't have to be specified, although type hints can be provided to improve performance. Belonging to the Lisp family of languages, it has a Lisp-style syntax that is characterized by its simplicity, expressiveness, and flexibility.

One of the key features of Clojure is its support for functional immutable data structures. These data structures cannot be modified once they have been created. This makes it easier to write correct, efficient, and concurrent programs in Clojure, as it eliminates the need for mutable state and the associated challenges of managing and coordinating access to shared state.

Despite being less popular, after 15 years since its release, Clojure has been successfully adopted by many companies and large organizations.

As a final note, it should be mentioned that Clojure follows Java's approach to remain backwards compatible. After 11 major releases, code is still compatible with version 1, which is essential for the maintainability and stability of a software project[2].

[1] `https://github.com/Tensegritics/ClojureDart`

2.1.2 Language Primer

Clojure has concise syntax and experimentation in the Read Eval Print Loop (REPL) is very common. Installation instructions can be found on the official site[2] and once installed the REPL can be launched with the `clojure` command.

As in many cases, we will start by showing a “Hello World” example in the REPL (Listing 1):

```
$ clojure
Clojure 1.11.1
user=> "Hello World"
"Hello World"
```

Listing 1: Hello World

Here, the Clojure REPL is started and the "Hello World" string is provided. Since everything is an expression, the same string "Hello World" is returned as it evaluates to itself. The prompt (user =>) indicates that we are in the user namespace. Namespaces in Clojure are equivalent to Java packages, but in this language, we can navigate between them and evaluate expressions without requiring the declaration of the fully qualified namespace path.
A more complex example would involve the creation and execution of a function, as shown in the following snippet (Listing 2):

```
(defn add-1 [x]
  (inc x))
#'user/add-1

(add-1 1)
==> 2
```

Listing 2: Function definition

Here, a function that accepts a single parameter (x) is defined. The definition returns a symbol that points to the created function, which we can then call and provide the argument 1.
It is noteworthy that the function arguments are defined using a vector ([x]), rather than a list as is more common in Lisp family of languages. This reduces the number of parentheses and makes the code more readable.
`To gain an appreciation for the significance of Clojure being hosted, we can examine the types of some primitives by utilizing the type function` (Listing 3):

```
user=> (type 1)
java.lang.Long
user=> (type "foo")
java.lang.String
user=> (type :foo)
clojure.lang.Keyword
user=> (type 'foo)
```

[2]https://clojure.org/guides/install_clojure

```
clojure.lang.Symbol
user=> (type (/ 1 3))
clojure.lang.Ratio
```

Listing 3: Primitives

We can observe that Clojure utilizes Java's numeric types for numbers (the same applies to Ints and Floats). It defaults to the higher precision numbers and will coerce to BigInt if necessary to avoid losing precision. This, along with the addition of the Ratio type, can be helpful in applications that require numeric computation and where precision is important.
One of the constructs that sets Clojure apart from other languages is its use of Persistent Immutable Data Structures [3]. While this comes at the expense of some overhead, these data structures offer operations that can be beneficial in many ways, ranging from simplicity to effective concurrency.
Below is a code snippet that demonstrates some of the most commonly used data structures (Listing 4):

```
user=> (type [1 2 3])
clojure.lang.PersistentVector
user=> (type {:a 1 :b 2})
clojure.lang.PersistentArrayMap
user=> (type #{1 2 3})
clojure.lang.PersistentHashSet
user=> (type '(1 2 3))
clojure.lang.PersistentList
```

Listing 4: Data Structures

All of these collections evaluate to themselves except for lists, which is the collection that Clojure code is structured with. This is why the list needs to be quoted (') to prevent evaluation.
We can see that there is a concise way of defining all the major collections with the help of the reader [4]. Usage of Sets is also quite common in Clojure as Sets are commonly used to check the presence.
Clojure developers prefer structuring code information using maps and vectors rather than classes, something that can result in less code. Additionally, those constructs can leverage many functions that are built into the core of the language.
The following snippet shows some common operations on data structures (Listing 5):

```
;; lines starting with selicolons are comments
;; get the item at index 1
(nth [1 2 3 4] 1)
2
;; same as above but now using the vector as a function instead of
;; function nth
user=> ([1 2 3 4] 1)
2
;; using nth will trigger the known in the Java world IndexOutOfBoundsException
```

[3] https://clojure.org/reference/data_structures
[4] https://clojure.org/reference/reader

```
user=> (nth [1 2 3 4] 5)
Execution error (IndexOutOfBoundsException) at user/eval167 (REPL:1).
null
;; using get will not
user=> (get [1 2 3 4 5] 5)
nil
;; using the 3-arity get function and providing a fallback
;; not the out of bounds case (-1)
user=> (get [1 2 3 4 5] 5 -1)
-1
;; using sets to check the presense
user=> (#{1 2 3 4} 3)
3
user=> (#{1 2 3 4} 5)
nil
```

Listing 5: Data Structure Operations

The final example shows what collection processing looks like (Listing 6):

```
user=> (apply + (map #(* 2 %) (filter odd? (range 10))))
50
;; we can write the same using the threading macro
;; to make the code more readable
user=> (->> (range 10)
(filter odd?)
(map #(* 2 %))
(apply +))
50
;; if we expand the threading macro we see that the
;; the code that is actually used if very similar to the
;; one we wrote initially
user=> (macroexpand '(->> (range 10)
(filter odd?)
(map #(* 2 %))
(apply +)))
(apply + (map (fn* [p1__2#] (* 2 p1__2#)) (filter odd? (range 10))))
```

Listing 6: Collection Processing

This final example effectively illustrates one of the core design principles of the Clojure language which is to remain compact while providing a wide range of built-in functions that users can easily leverage. This allows users to create their own functions that are not limited to specific types defined previously, nor do they share the same complexity of parameterized data types seen in languages such as Java or Scala.

2.1.3 REPL-driven development

Clojure is a programming language that is well-suited for web development, and it offers a number of features and tools that can help to make the development process more efficient and effective. One such feature is its REPL-driven development, which allows developers to make changes to their code, evaluate it and see the results in real time without having to restart the web server.
The REPL-driven development can save a lot of time and effort, as it allows the developer to iterate quickly and make changes to the code without having to interrupt their workflow. It is one of the many features of Clojure that makes it a popular choice for web development.

2.1.4 Benefits

There are several benefits that come with choosing Clojure as the implementation language for this project.
Clojure code tends to be smaller than implementations in other languages, particularly those in the ALGOL family such as Java or C#. Smaller code often means less development time and fewer bugs.
Since this project involves music processing and evolutionary algorithms, an expressive language with strong data processing functionality like Clojure is highly advantageous.
REPL-driven development facilitates experimentation, which saves time and prevents unnecessary server restarts.
Clojure is widely used in web development, so there are many mature libraries available for working with web servers and databases.
Lastly, the development principles of the language and the community strive to avoid breaking changes. This saves time and frustration for a project that requires a significant amount of time to develop and evolve.

2.2 Design decisions and considerations

Evolduo is a small project with less than 5K lines of Clojure code. The choice of Clojure helps keep a relatively small codebase while enabling users to perform a variety of tasks and allowing flexibility for future growth.
When designing a project, there are many choices to make, depending on factors such as time constraints, scope, and personal preferences.
It's easy to get overwhelmed by the numerous options available, ranging from software architectural patterns (e.g., CQRS vs. MVC), databases (e.g., XTDB[5] or Datomic[6] vs. traditional SQL databases), frontend design approaches (SPAs vs. Server-Side Rendered pages), and code structure (microservices vs. monolithic), among others.
Even after architectural choices have been made, engineers must still choose specific libraries and tools to materialize the design choices. Factors like complexity, feature set, and production readiness come into play here. In many cases, engineers will select the options that are most familiar to them, feel easiest to use, are stable and mature.
When it comes to infrastructure, there are still many different options to consider. Should the application

[5]`https://xtdb.com/`
[6]`https://www.datomic.com/`

be deployed to a Platform as a Service (PaaS[7]) such as Heroku[8] to minimize infrastructure knowledge and maintenance costs? Should it be deployed to a small EC2 AWS instance via CloudFormation? Should it be packaged as a Docker container and deployed to a self-managed Virtual Private Server (VPS)? The number of choices goes on and on. In many cases, ease and simplicity come at a monetary cost, which may not be desirable.
There are also many choices to make regarding security, performance tracking, analytics, emails, and more.
The selection of tools and libraries for Evolduo is optimized based on the author's knowledge, time availability, user privacy orientation, and the intent to minimize operational expenses as much as possible. This information should help readers understand the reasoning behind the design choices that are explained in the following sections.

2.3 Initial thoughts and prototyping

As mentioned earlier, some of the choices for tools and libraries can be subjective and depend on the developer's personal preferences. However, there are some design decisions that may be less clear and require more careful consideration. In the following section2.3, we will discuss several key design choices related to music representation, data storage and application modularization.

2.3.1 Music representation

This passage discusses the considerations and decisions made in selecting a music representation format for the Evolduo. The format needed to be both expressive and simple, with the ability to programmatically convert MIDI pitch values to the chosen library's format. Additionally, the format needed to have a JavaScript implementation that would allow interactive playback in the user's browser without requiring any additional software downloads or installations.
After considering various options, it was decided that the ABC[9] notation format was the best fit for Evolduo's needs due to its simplicity and feature set. The JavaScript library ABC.js[10] is capable of interpreting ABC notation in the browser, as well as allowing users to listen to and download the results as MIDI and WAV files. MIDI was deemed essential as it allows the processing of music with compatible software, such as musical composition software like MuseScore[11] or Digital Audio Workstations (DAWs) like Ardour[12]. Some of the music produced by Evolduo was rendered using VST instruments and included in the samples[13] section of the project's website.

2.3.2 Database

For the size and complexity of this project, SQLite was the initial choice. The convenience of the underlying database being a single file is hard to beat, especially in the prototyping phase of a project. Moreover, SQLite comes with a feature set that covers the needs of the project. Despite basic needs, SQLite supports

[7]`https://en.wikipedia.org/wiki/Platform_as_a_service`
[8]`https://www.heroku.com/`
[9]`https://abcnotation.com/`
[10]`https://www.abcjs.net/`
[11]`https://musescore.org/en`
[12]`https://ardour.org/`
[13]`https://evolduo.cons.gr/samples`

advanced features such as JSON functions, Views, and derived columns.
After the initial prototyping phase and as soon as this project was deployed to a VPS, which happened in mid-June 2022, SQLite was dropped and replaced with PostgreSQL.
There are several reasons behind this transition.
The first reason has to do with the supported data types and the flexibility of mapping them to JVM types. PostgreSQL supports more data types, such as dates, which are not as feature full in the SQLite case. For example, inspecting human-readable dates in the database (in PostgreSQL's case) is far more convenient compared to Unix epoch times.
The second reason relates to the transparency of the data conversion from and to the database data types. Some data types, such as booleans and JSON, had to be manually converted, which is cumbersome and error-prone.
An additional reason involves the convenience of accessing and inspecting the database via Secure Shell (SSH) tunneling[14], which is essential for the majority of software projects.
The last reason has to do with the fact that PostgreSQL is a database that was already installed on the server where Evolduo was deployed, removing the burden and cost of having to deal with it.

2.3.3 Application modularization

Evolduo can be viewed as two separate logical components: Firstly web platform that enables users to perform a set of actions, and secondly, the creation of music using evolutionary algorithms.
An initial design consideration was the potential separation of the web platform and the music processing. The main reason behind this reasoning was the fact that web frameworks, such as Django[15], which is coded in Python, provide built-in features such as an administration interface, libraries for user and permission management, and email sending capabilities. All of these functionalities would be needed for Evolduo, and avoiding redundant development tasks is a key aspect of software engineering. To support this selection, all data would be stored in a Django-compatible database (such as PostgreSQL). Evolduo, which in this case would exist as a standalone background task, retrieves the data from the database or some communication bus, process it, and returns it back to its source.
This was a compelling design consideration for several reasons. It felt like existing tools would be used efficiently, and the main focus would be to design a system that generates music and not creating another user management application, which would have bugs, be less secure, and lack necessary features.
The drawback of this approach was that splitting a single system into two parts would introduce several undesired side-effects. These drawbacks include the challenges of maintaining two different systems, switching between different programming languages, dealing with more complex deployment processes, and establishing a communication pattern between the two components. This could be achieved in several ways, ranging from a quick and simplistic method like directly accessing and writing to the same database, to a more organized, such as accessing and writing to the same database directly, to a more structured approach, such as having a dedicated messaging queue.
The first option was ruled out because it's a bad architectural pattern that can lead to confusion and data inconsistencies. As a general rule, only one system should be responsible for writing to the database at any given time. Starting a brand new system with such a violation was not a compelling prospect, and the required investment would be too big to abandon this design decision later.

[14] `https://www.ssh.com/academy/ssh/tunneling`
[15] `https://www.djangoproject.com/`

The second option would involve incorporating and maintaining a messaging bus or queue. That could be something like RabbitMQ[16], but it would require the installation and maintenance of an additional component, which seemed excessive for this project. The most appealing alternative was Redis, since it was already installed on the VPS where Evolduo was deployed, and it already supported a messaging queue[17]. Thankfully, this possibility reached a dead end, as the most popular Redis Clojure library[18] used a custom format that was not natively supported in Python. Consequently, this potential approach was abandoned, leading to the decision to develop the entire system in Clojure, with the idea that maintaining a single system, with which the author was most familiar, would be more cost and time-effective, even though it would mean that the user management part would need to be written from scratch.

2.4 Application Architecture

As explained previously, Evolduo can be viewed as two different components: a web platform and a music engine.
In this section, we will be focusing on the former aspect and the corresponding architectural decisions. Everything that is music related with be covered in the next chapter3.
Evolduo follows the widely adopted Model-View-Controller (MVC) software architectural pattern which is commonly used in many well-known application frameworks like Ruby on Rails, Django and Spring Framework.

2.4.1 Model-View-Controller (MVC)

Even though Clojure does support class-like constructs[19], vectors and maps are typically used to organize information.
The Model portion of the project contains SQL query abstractions, rather than definitions of domain models that correspond to database entities.
The View functions accept data and produce a representation of that information, such as HTML markup or JSON objects.
The Controllers are responsible for processing HTTP requests, handling user input validation and permission checking, selecting the appropriate View and Model data, and returning the appropriate HTTP status codes with corresponding information.

2.4.2 Entity-relationship model

Figure 1 presents the Entity-relationship Diagram (ERD) of Evolduo.
The main entities of the project are Users and Evolutions. The majority of relationships and actions are oriented around those two entities.
The relationship between them is that users create evolutions, which contain a number of iterations. Each iteration has a number of chromosomes representing musical phrases.
Within Evolduo, users can invite others to participate in private evolutions, and they also have the option to submit ratings for the chromosomes they have access to.

[16]`https://www.rabbitmq.com/`
[17]`https://redis.com/redis-best-practices/communication-patterns/event-queue/`
[18]`https://github.com/ptaoussanis/carmine`
[19]https://clojure.org/reference/datatypes#_deftype_and_defrecord

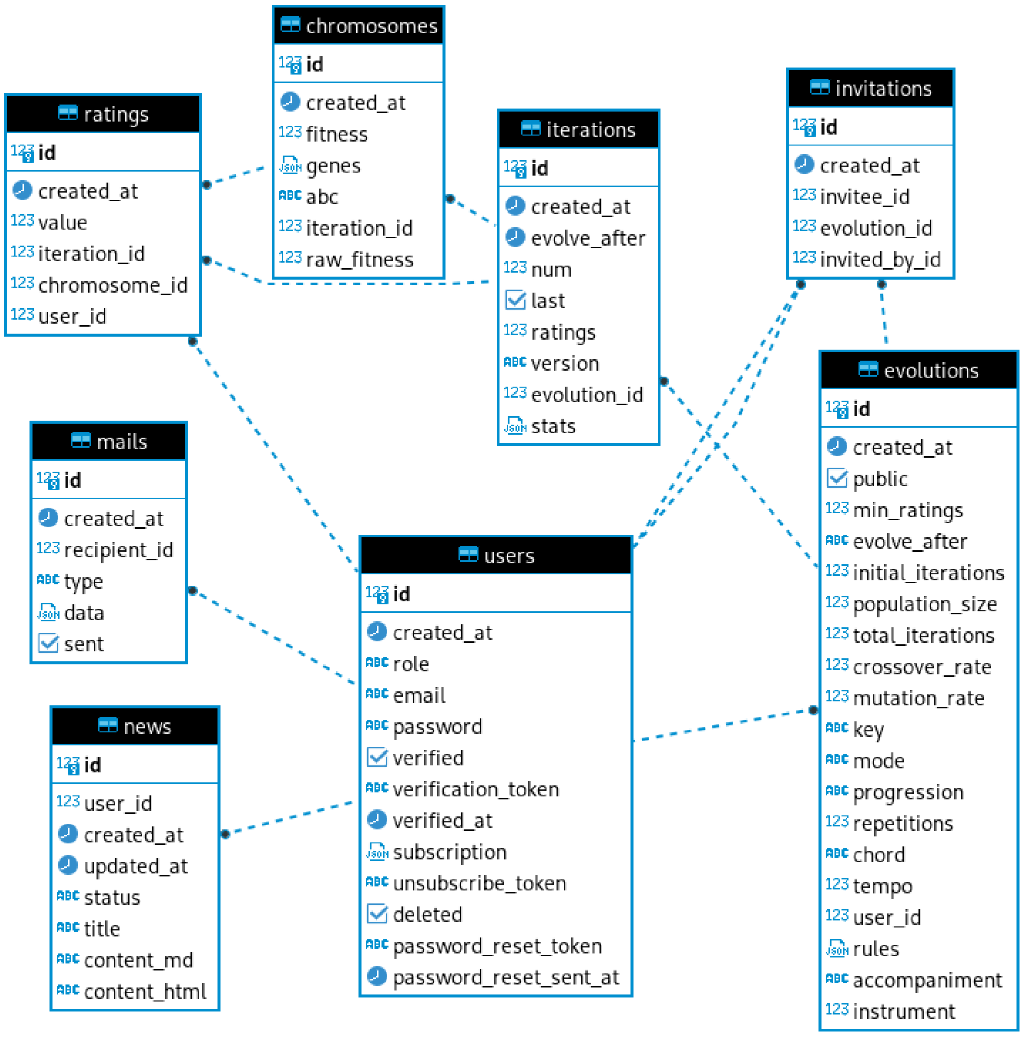


Figure 1: Database Schema

Additionally, a News entity was implemented for creating announcements that could potentially trigger notifications for users. However, no news posts have been made at the time of writing this thesis[20].

2.4.3 Styling and interactivity

The application's styling is done using the Bulma CSS framework, which provides all the necessary components for a standard web application.
The use of JavaScript is limited to integrating ABC.js, which allows users to see a rendered score for an ABC track and listen to the tracks in the browser.

[20]`https://evolduo.cons.gr/news`

2.4.4 User Actions and Management

Users can register for an account using their email address and password, and can subsequently log in and request a password reset if necessary. However, at the time of writing, users are not able to change their email or password. The implementation of these features was postponed in favor of adding more customization options for music generation.
In terms of music-related features, Evolduo enables users to create Evolutions, which are processes that evolve musical phrases. Additionally, users have the capability to monitor all the generated phrases across various generations. Users can invite others to participate in private evolutions that are not visible to other or anonymous users. Furthermore, users have the ability to rate tracks, which serves as a means to influence the evaluation of music tracks.
Regarding user data privacy, the application was designed in compliance with GDPR regulations, and users are allowed to delete their profiles.

2.5 GDPR

The General Data Protection Regulation (GDPR) is a regulation of the European Union (EU) that establishes a set of rules and standards for the collection, use, and storage of personal data. It is designed to protect the privacy of individuals and give them greater control over their personal information.
The GDPR applies to any organization, whether based in the EU or not, that collects or processes the personal data of EU residents. This means that companies that operate in the EU, or that have customers in the EU, must comply with the GDPR.
The GDPR sets out a number of rights for individuals, including the right to access their personal data, the right to have their personal data erased, and the right to object to the processing of their personal data. It also imposes a number of obligations on organizations that collect or process personal data, such as the obligation to obtain consent from individuals before collecting their personal data, and the obligation to keep personal data secure.
The GDPR is considered an essential regulation because it helps to protect the privacy and personal data of individuals, and it provides a common set of rules and standards for organizations that collect and process personal data. This helps to ensure that personal data is handled in a fair and transparent way, and that individuals have greater control over their own information.
Evolduo's data is stored in a VPS that is located in France and the majority of its userbase is located in EU. As such Evolduo must comply with GDPR regulations.
Evolduo allows users to delete their accounts which is one of the most important points of GDPR regulations. Account deletion anonymizes all Personally Identifiable Information (PII), which is just the email, and keeps the data the user has generated as it cannot be linked back to the original user.

2.6 Security and Privacy

Building secure and privacy-oriented software systems is one of the most challenging tasks in software engineering. System security often comes at the cost of user privacy or financial resources. In the case of Evolduo, which is a small project with limited resources, options such as running the application on a Platform as a Service (PAAS) provider like Heroku or using a DDoS Firewall like `Cloudflare`[21] are

[21] `https://www.cloudflare.com/`

not feasible. Instead, Evolduo runs on a VPS, and various proactive measures have been implemented to protect the platform and user information.
This section highlights some of the security and privacy decisions that have been made, potential alternatives, and why certain options were not chosen.
While Single Sign-On (SSO) is considered both convenient and secure, using a third-party provider such as Twitter is not a viable option for Evolduo. Although platforms like Twitter are less prone to attacks compared to a small software system like Evolduo, using a third-party SSO provider often entails extensive tracking of users, which conflicts with the goal of protecting user privacy.
To prevent bots and users from abusing the site, Evolduo utilizes a Firewall (as described in section 2.6.1) and a CAPTCHA mechanism (as explained in section 2.6.2).
Using well-known analytics platforms like Google Analytics has privacy implications, which is why those were not integrated into the application. During the prototyping phase, a small self-hosted alternative was used[22] but it was quickly dropped due to GDPR requirements, which mandate that users must give their consent before their visits can be tracked.
The only third-party platforms that Evolduo uses are Sentry [23] for tracking errors and Mailjet [24] for sending emails. In the case of Sentry, no personally identifiable information (PII) is tracked. However, it is necessary to provide users' email addresses to Mailjet for sending emails.
Finally, resource serving (Javascript and CSS) is done through Nginx, as utilizing Content Delivery Network (CDN) services often results in them being identified as trackers by privacy-focused tools like Privacy Badger.

2.6.1 Firewall

Evolduo employs the `fail2ban`[25] intrusion prevention software framework, a widely available security mechanism across major Linux distributions.
This software runs as a `systemd`[26] service, monitoring log files for malevolent indicators, including excessive password failures and exploits. Fail2ban serves as a firewall for Evolduo, capable of blacklisting IP addresses upon detection of repetitive failed login or signup attempts.
The configuration directive for Evolduo in `/etc/fail2ban/jail.d/jail.local` is present in Listing 7:

```
[evolduo]

enabled = true
port = https
filter = evolduo
logpath = /var/lib/docker/containers/*/*-json.log
maxretry = 20
findtime = 120
bantime = 600
```

Listing 7: Evolduo Jail

[22]https://github.com/milesmcc/shynet
[23]http://sentry.io/
[24]https://www.mailjet.com/
[25]https://www.fail2ban.org
[26]https://systemd.io/

This configuration enables the parsing of docker container logs and allows for 20 login or signup attempts within a 120 second window. If this limit is exceeded, a 600 second ban will be imposed on the offender. The filter definition for this jail file is shown in Listing 8:

```
[Definition]
failregex = .*Invalid (login|signup) attempt from <HOST>.*
ignoreregex =
```

Listing 8: Evolduo Jail Filter

Invalid login and signup attempts generate warning log entries that match the fail regex pattern. These entries are tracked using the `<HOST>` field, which represents the IP address of the user.
For this configuration to work it's important to have Nginx proxy the header of the real `IP` of the user, which is the `$remote_addr` and will be provided as the `X-Forwarded-For` header (Listing 9):

```
[Definition]
location / {
                proxy_pass      http://10.19.0.5:4000;
                proxy_set_header X-Forwarded-For $remote_addr;
           }
```

Listing 9: Evolduo Nginx Proxy

Fail2ban offers a client that allows checking the status of the different jails configured. It is worth noting that the container has been renamed to `foo` to minimize the output (Listing 10):

```
$ fail2ban-client status evolduo
Status for the jail: evolduo
|- Filter
|  |- Currently failed: 1
|  |- Total failed: 1
|  `- File list: /var/lib/docker/containers/foo/foo-json.log
`- Actions
   |- Currently banned: 0
   |- Total banned: 1
   `- Banned IP list:
```

Listing 10: Evolduo Fail2ban Client

An important consideration when utilizing fail2ban with docker is that redeployments of the application will result in a new container, necessitating a restart of the fail2ban service to track the newly created log files. This issue has been discussed in detail on GitHub[27].

2.6.2 Captcha

As with the majority of applications where users can register and perform certain sensitive actions, such as triggering the sending of emails, the use of CAPTCHA[28] is necessary. CAPTCHA is a type of challenge-

[27]https://github.com/fail2ban/fail2ban/issues/2947#issuecomment-784229259
[28]acronym that stands for Completely Automated Public Turing test to tell Computers and Humans Apart

response test used in computing to determine whether or not the user is human. This is often used to prevent automated programs, or bots, from accessing certain websites or services. Well-known CAPTCHA services include reCAPTCHA`https://developers.google.com/recaptcha/` and hCaptcha[29].
The initial CAPTCHA in Evolduo, developed during the prototyping phase, was an esoteric music challenge that would ask the user which was the quadrant (also known as tetrad [30]) chord at the nth degree of a given key and scale. For example, if the user was given C major and the 2nd degree, the answer would be Dm7. While this approach seemed entertaining for this sort of application, it would drastically put off users from signing up.
Per the suggestions of friends during development, this was quickly changed to the use of nanocaptcha[31], which is a simple self-contained library for generating CAPTCHA challenges that are more familiar to end-users.
The reason for not selecting well-known CAPTCHA services was primarly user privacy.

2.7 Error tracking and performance

Measuring and optimizing performance is a task that can be approached in many different ways. When developing a system that is aimed to be used by end-users, providing a good user experience is a very important aspect. Even though the end goal is clear, taking the appropriate measures is usually not as simple.
The main two aspects to consider for this problem are error handling and performance tracking.
Error tracking is simpler as it's just monitoring events of errors that were triggered by the system. As soon as this happens, all that's left to do is to identify the root cause and patch the system.
Error reporting is done via Sentry, which will provide a stack trace of the error as well as context information (operating system, browser, system version, etc.). Sentry needs to be monitored regularly, and it provides a weekly report regarding the errors that occurred on the platform.
Evolduo is trivial performance-wise from the web activity standpoint. It is expected that it will have just a handful of active users, even if it develops a small user base. On the other hand, the background evolutionary processes are expensive tasks in terms of both memory and CPU usage. Those will process and generate an important amount of data for each Evolution iteration. The next chapter will make it clearer what exactly this process contains.
The quote "Premature optimization is the root of all evil" applies to this work too. Trying to guess and optimize the platform for the best results is a guessing game and it can involve an important amount of work that could be otherwise invested in something more useful.
Performance tracking is also done via Sentry, which provides information on how much time an HTTP request took to finish. This is useful as it can reveal poorly optimized code or missing database indexes. What Sentry is not monitoring right now are background processing tasks, such as evolving an evolution's iteration. This is something that is left for future improvements.

[29] `https://www.hcaptcha.com/`
[30] `https://en.wikipedia.org/wiki/Tetrad_(music)`
[31] `https://github.com/logicsquad/nanocaptcha`

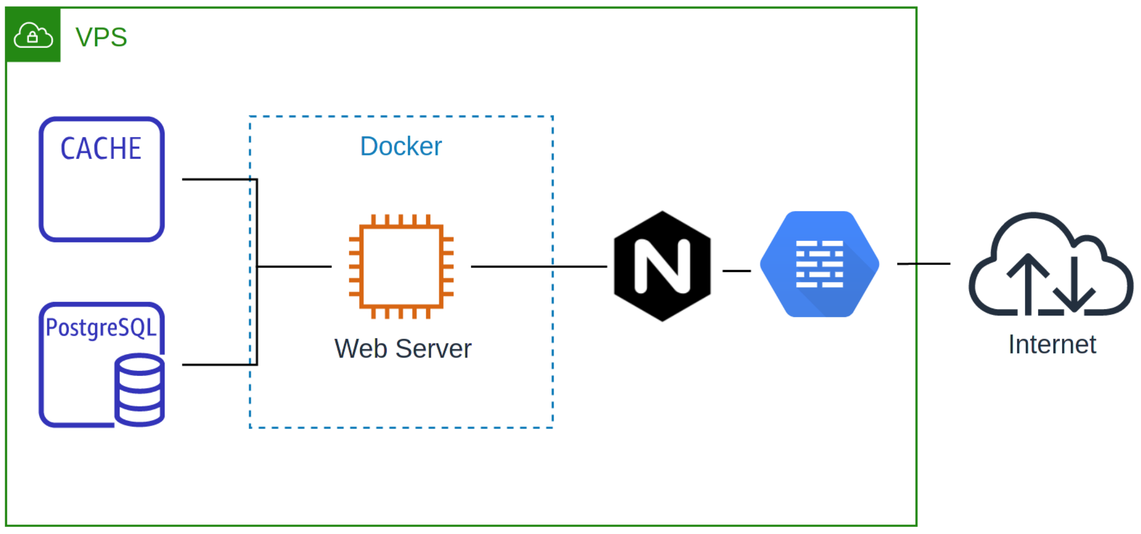


Figure 2: Infrastructure diagram

2.8 Development and Deployment

2.8.1 Development

Evolduo is a monolithic web application, and as such, its infrastructure needs are limited. The entire infrastructure can be seen in Figure 2. As already described in Section 2.3.3, for the size and complexity of this project, utilizing a microservice or multiservice architecture would be overkill.
Evolduo's source code is hosted on Github[32], and it is using Github actions for its build and deployment tasks.
There are two tasks this project is using. One runs tests, and it is triggered on each commit push. The second one prepares a release when a git tag is added.
The steps included in the release task are:

1. Compile the project.
2. Package it as a JAR file, which is a common practice for JVM-based projects.
3. Create a Docker image including the newly built JAR file.
4. Push the image to the Docker Hub[33].

All these actions are publicly visible and are utilizing the free tiers provided by those services.
One could argue that even Dockerizing the build is unnecessary complexity, but the main reason behind this is that Evolduo is running on a shared VPS, which also runs other services. Docker provides resource isolation, convenience (there may be different needs for JVM runtimes), and security. If a service is compromised, the others are running in different isolated environments.

[32]`https://github.com/kongeor/evolduo-app`
[33]`https://hub.docker.com/repository/docker/kongeor/evolduo-app`

The development of this project, given its complexity and the fact that a single person is working on it, does not require a complex development lifecycle (e.g., ticket management, working on different branches, or creating PRs). The majority of development is done on the main branch, and most of the commit pushes to the main branch are also released to the production environment.

2.8.2 Deployment

Deploying Evolduo is a simple and mostly manual process. It involves the following steps:

1. Connect to VPS via SSH.
2. Pull the source code as it may have updated static resources.
3. Bump the version in Docker compose file to reflect the newly built version.
4. Restart the docker container.

With Evolduo having just one instance, a deployment will cause a small outage (Nginx returns HTTP 502 responses in that case) and this is lasting a couple of minutes. For the amount of traffic this project has this issue is negligible.

2.9 Libraries and tools

Evolduo is using 27 third-party libraries, most of which are very common for web-based projects. In this subsection three of them will be described as their functionality is important and different when compared to other JVM or OOP languages. A fourth one, the one related to evolutionary algorithms, will be described in the next chapter.
The three libraries that will be described here cover the needs for for a data-driven architecture (`integrant`[34]), also known in the industry as dependency injection, SQL queries (`honeysql`[35]), and schema validation (`malli`[36]).

2.9.1 Integrant

Integrant is a Clojure micro-framework for building applications with a data-driven architecture. It serves the same purpose as the Inversion of Control (IoC) container in Spring Framework. However, instead of using class based annotations the dependency graph is constructed using a Clojure map, as can be seen in Listing 11:

```
(def config
  {:adapter/jetty           {:handler  (ig/ref :handler/run-app)
                             :settings (ig/ref :config/settings)}
   :handler/run-app         {:db       (ig/ref :database.sql/connection)
                             :settings (ig/ref :config/settings)}
   :database.sql/connection {:settings (ig/ref :config/settings)}
   :database.sql/migrations {:settings (ig/ref :config/settings)}
   :config/settings         {}
```

[34] `https://github.com/weavejester/integrant`
[35] `https://github.com/seancorfield/honeysql`
[36] `https://github.com/metosin/malli`

```
  :evolution/timer         {:db       (ig/ref :database.sql/connection)
                            :settings (ig/ref :config/settings)}
  :mail/timer              {:db       (ig/ref :database.sql/connection)
                            :settings (ig/ref :config/settings)}})
```

Listing 11: Evolduo System Config

Here `:database.sql/connection` configuration depends on `:settings` which contains the database credentials and the `:handler/run-app` which configures the web server routes and settings, depends on `:database.sql/connection` etc. This is a very concise way to describe the entire system in one place, making it easy to reason about, modify and extend.

2.9.2 Honeysql

There are a number of libraries in the Clojure ecosystem that cover the need for querying SQL databases in a convenient way. These range from relatively thin SQL libraries (e.g. HugSQL[37], YeSQL[38]) to more traditional like Object-Relational Mapper (ORM) abstractions (e.g. Toucan[39]).
Honeysql falls on the thiner side, but instead of using SQL files like hugsql, it uses Clojure's data structures to represent SQL queries. In Listing 12 it is presented how a data structure is transformed to a vector of query and variables that can then be passed to the query engine:

```
(let [q-sqlmap {:select [[[:raw "count(*)"] :count]]
                :from [[:evolutions :e]]
                :where
                [:and
                 [:> :e.created_at [:raw ["now() - interval '1 day'"]]]
                 [:= :e.user_id 1]]}]
  (h/format q-sqlmap))
=>
["SELECT count(*) AS count
  FROM evolutions AS e
  WHERE (e.created_at > now() - interval '1 day') AND (e.user_id = ?)" 1]
```

Listing 12: Honeysql example

2.9.3 Malli

Malli is a library that can validate data against predefined schemas, among other things. In Evolduo it serves a dual purpose: ensuring that no bad data are stored in the database and providing form validation errors with human-readable messages.
Listing 13 shows a schema that validates user signup data:

```
;; Minimum eight characters, at least one uppercase letter,
;; one lowercase letter, one number and one special character
```

[37] https://www.hugsql.org/
[38] https://github.com/krisajenkins/yesql
[39] https://github.com/metabase/toucan

```
(def password-regex #"^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9]).{8,}$")

(defn safe-lower-case [s]
  (when s
    (str/lower-case s)))

(def Signup
  [:and
   [:map {:closed true}
    [:email {:decode/string {:enter safe-lower-case}}
     [:re {:error/message "invalid email"} email-regex]]
    [:password [:re {:error/message "Invalid password"} password-regex]]
    [:password_confirmation [:string]]
    [:captcha [:string {:min 1}]]
    [:newsletters {:optional true} [:string]]]
   [:fn {:error/message "passwords must match"
         :error/path [:password_confirmation]}
    (fn [{:keys [password password_confirmation]}]
          (= password password_confirmation))]])

(me/humanize (m/explain Signup {:email                 "foo@examplecom"
                                :password              "Foo123456"
                                :password_confirmation ""}))
=> {:email ["invalid email"]
    :captcha ["missing required key"]
    :password_confirmation ["passwords must match"]}
```

Listing 13: Malli example

Here, ordinary Clojure constructs like functions and maps are used to define a schema with the following characteristics:

- It’s closed, which means that not known keys (e.g. `:name`) would trigger an error.
- It shows how to use regex validation and provide custom error messages instead of the built-in ones (with the use of `:error/message`).
- It demonstrates how to use functions for more complex validation, like checking that two fields have the same value.
- It shows how, with the use of standard Malli functions, we can construct a map that can be conveniently passed to other parts of the system (e.g., view layer to display the errors).

It should be noted that in this example namespace imports have been omitted for brevity.

2.10 Infrastructure

As described previously Evolduo runs in a docker container in a small VPS behind an Nginx server.

2.10.1 Nginx and HTTPs

Nginx is one of the most commonly used web servers and people seem to prefer it for its speed and security. Nginx is lightweight and is present on all major Linux distributions.
As in the majority of cases, here Nginx is proxying requests to the Docker container (Diagram2). This allows two things to be enabled. One is having Let's Encrypt configured at the system/Nginx level, which is essential for all web applications at present. Let's Encrypt provides a valid HTTPs certificate, ensuring that all traffic is served over a secure channel. The second feature that Nginx provides is serving static assets (such as JS, CSS, and audio files), which it can do much more efficiently than a JVM runtime.

2.10.2 Databases

Evolduo is utilizing two databases, PostgreSQL as its main storage and Redis for session cache. The databases are not running inside Docker as this can potentially have unwanted consequences when it comes to management and versioning. It is considered a much safer option to utilize a dedicated database instance for these cases.
Redis allows the web server to be redeployed without losing user sessions. User sessions last for 30 days, which means that users will be logged into the application without having to re-login for a month. Kicking out users on each deployment would provide a bad user experience and could drive away the already small user base the project has. Additionally, at the time of the release, the project didn't have a password recovery functionality, which could be a dead end for users who forgot their password. However, the password reset functionality was implemented shortly after the project was announced4.1.

2.11 License

Evolduo's source code is released under the Affero General Public License, and the music tracks created by the project belong to the public domain. This section explains the motivation behind this dual license selection.
The Affero General Public License (AGPL) is a type of open-source license that is designed specifically for software that is used over a network, such as a web-based application. It is based on the GNU General Public License (GPL), but it includes an additional provision that requires the source code of the software to be made available to users who interact with it over a network.
The AGPL is considered to be a good candidate for a SaaS (Software as a Service) project because it ensures that users of the software have access to the source code, even if they are not directly downloading and running the software on their own computers.
In addition to its network-oriented provision, the AGPL has many of the same features as the GPL, including the requirement that the source code of the software must be made available to anyone who receives a copy of the software, and the requirement that any modifications to the software must also be made available under the AGPL. These provisions ensure that the software remains open and accessible, and that any improvements made to the software are shared with the broader community.
The public domain is important for generated content because it provides a way for the data to be freely available to others without restriction. When a work is in the public domain, it is not protected by copyright or other intellectual property laws, and anyone can use, modify, distribute, or sell the work without obtaining permission from the creator.

For generated content, such as computer-generated art, music, or writing, the public domain can be particularly important because it allows others to build upon and expand the original work, creating new and innovative creations. This can foster creativity and collaboration within the community of creators, and it can help to ensure that new works are not held back by restrictive intellectual property laws.

Chapter 3: Evolutionary Music

3.1 Evolutionary algorithms

An evolutionary algorithm is a type of optimization algorithm that uses techniques inspired by evolutionary biology to find solutions to problems. These algorithms are typically used to solve problems that are difficult or impossible to solve using traditional algorithms, such as problems with a large search space or complex constraints.
The basic idea behind evolutionary algorithms is to create a population of potential solutions to a problem, and then use principles inspired by natural selection, reproduction, and genetic mutation to evolve the population over time, with the goal of finding the best possible solution to the problem. This involves evaluating the fitness of each solution in the population and using this information to guide the evolution of the population.
John Holland and David Goldberg are both influential figures in the field and they have made many important contributions to the development of evolutionary algorithms[3, 4].

3.2 EvolTrio

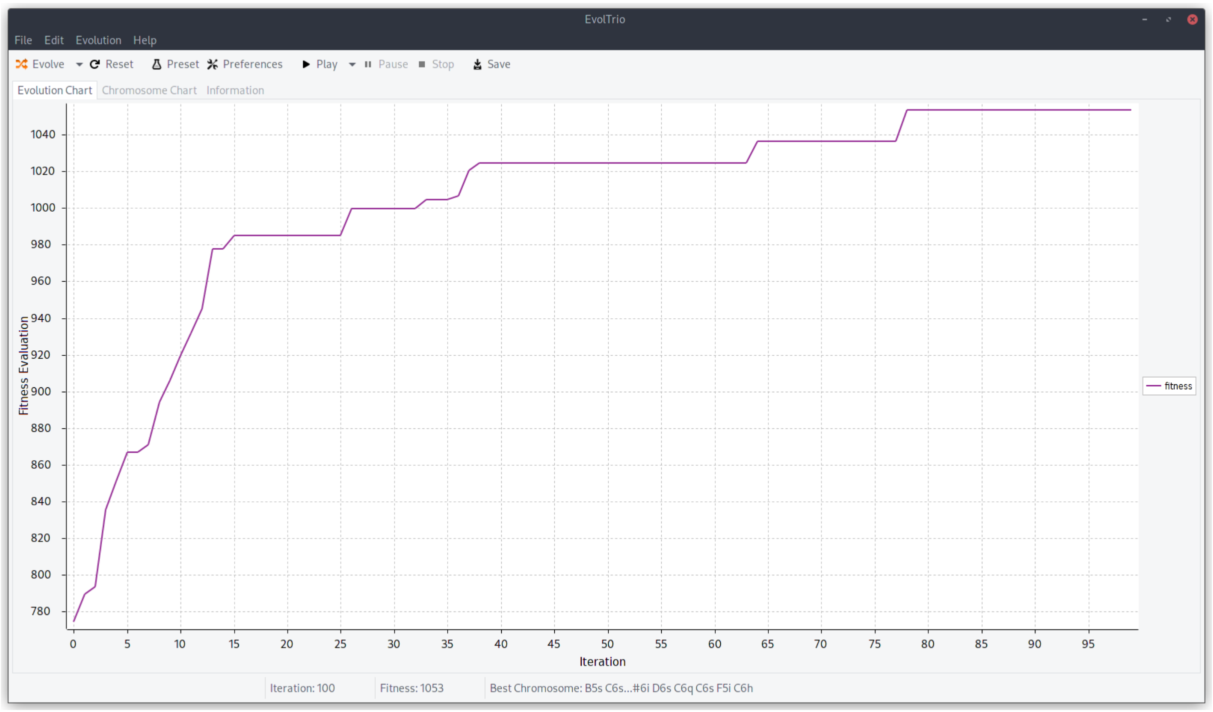
Author's undergraduate work involved the development of a similar program: A program that would compose musical phrases using evolutionary algorithms[5].
At late 2000's, software was mainly distributed as desktop applications, and EvolTrio followed suit. This application was written, and rewritten a number of times, mainly to change the GUI libraries. Initially it used the Java FX that eventually became obsolete and dropped, it was rewritten with the use of Apache Pivot, and its final form used the SWT which was developed and used by the Eclipse foundation in a number of application with the Eclipse IDE being probably one of the most well known one.
EvolTrio also used the Java Web Start launching mechanism which would make it easier for users to use the app. EvolTrio intended to use a web service that would store user feedback submissions but this work was never finished.

3.3 Chickn

EvolTrio used the JGAP java library for the genetic algorithms, one of the most mature and well known libraries in the Java ecosystem. The same library could be used in Evolduo with the use of Java interop, but the implementation would still be verbose.
A few years before the work on this thesis, the Chickn library was developed by the author. Chickn is a genetic algorithms[40] library for Clojure. During the Evolduo work, the library was improved, taking a lot of inspiration from Sean Moriarity's book, Genetic Algorithms in Elixir[6].
The following code shows how to define a problem where the algorithm will try to find a chromosome with all genes being 1 (Listing 14):

```clojure
(ns chickn.examples.hello-world
  (:require [chickn.core :refer [default-cfg init-and-evolve higher-is-better]]
            [chickn.util :as util]))
```

[40]It should be noted the terms Evolutionary algorithms and Genetic algorithms are used interchangeably in this project, even though those are not technically the same.



Iteration: 100 Fitness: 1053 Best Chromosome: B5s C6s...#6i D6s C6q C6s F5i C6h

Figure 3: EvolTrio main view

```
(def one-or-zero (fn [& _] (if (> (rand) 0.5) 1 0)))

(def population-size 20)

(def chromo-gen #(repeatedly population-size one-or-zero))

(defn fitness [xs]
  (apply + xs))

(defn solved? [_ {:keys [best-chromosome]}]
  (every? #(= 1 %) best-chromosome))

(def mutation-op
  #:chickn.mutation
          {:type          :chickn.mutation/rand-mutation
           :rate          0.3
           :random-func   rand
           :mutation-func one-or-zero})

(def config (merge
              default-cfg
              #:chickn.core
                      {:chromo-gen chromo-gen
                       :fitness    fitness
                       :solved?    solved?
```

```
                          :reporter   util/noop
                          :mutation   mutation-op
                          :comparator higher-is-better}))
```

Listing 14: Chickn Example

If we provide this configuration to the Chickn process we will immeditaly get a solution to this problem:

```
(dissoc
  (init-and-evolve config 100) :population)
=> {:solved? true, :iteration 7,
    :best-chromosome [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1],
    :time 1}
```

Listing 15: Chickn Example Cont.

The development of Evolduo helped evolving and improving the Chickn library and the Chickn library itself, proved to be a helpful and convenient library that helped the development of Evolduo.

3.4 Music

3.4.1 Music theory basics

Music theory is the study of the language and notation of music. It encompasses a wide range of topics, including the principles of melody, harmony, and rhythm, as well as the structure and function of intervals, chords and scales.
Pitch is the perceived highness or lowness of a sound. It is determined by the frequency of the sound waves. Higher-pitched sounds have a higher frequency, while lower-pitched sounds have a lower frequency.
An interval is the distance between two pitches, and it is usually measured in half steps or whole steps. For example, the interval between C and D is a whole step, while the interval between C and E is a major third (two whole steps).
One of the fundamental concepts in music theory is the idea of a musical scale, which is a series of pitches arranged in a specific order.
The most common scale in Western music is the major scale, which consists of eight pitches and has a bright, happy sound. The pitches in a major scale are arranged in a specific pattern of whole and half steps (W-W-H-W-W-W-H), and this pattern is what gives the scale its unique sound.
Another important concept in music theory is the idea of a chord, which is a group of three or more pitches played at the same time. Chords are an essential element of harmony, which is the way that chords are used in music to create a sense of unity and movement. There are many different types of chords, including major chords, minor chords, and diminished chords, each of which has a unique sound and function in music.
Chord progressions are sequences of chords that are played in a specific order. Chord progressions can create a sense of movement and direction in music.

The root note is the first note of a chord or scale, and it is the pitch around which the chord or scale is built. For example, the C major scale consists of the notes C, D, E, F, G, A, and B, and the root note of this scale is C.
A mode is a type of musical scale and it is derived from the major scale. Different modes have different sounds and are used in music in different ways to create variety or different feelings.
Overall, music theory is a vast and complex subject that encompasses many different principles and concepts.

3.4.2 ABC notation

ABC notation is a shorthand musical notation system using letters, numbers, and symbols, making it easier to share and communicate musical pieces in written form.
In ABC notation, each line of music is divided into small, easy-to-read sections called bars. Each bar is written as a string of characters, with each character representing a specific element of the musical notation. For example, the letter A represents a musical note, and the symbol "," represents a rest.
Here is an example of ABC notation:

```
X:1
T: The Star of the County Down
M: 6/8
L: 1/8
R: Jig
K: Dmaj
|: D2D DFA | G2G GFG | A2A AFA | B2B Bcd |
d2d dcB | A2A AFA | G2G GFG | E2E E2D :|
|: B2B BAF | G2G G2E | F2F FEF | E2E E2D |
D2D DFA | G2G GFG | A2A AFA | B2B Bcd |
d2d dcB | A2A AFA | G2G GFG | E2E E2D :|
```

Listing 16: ABC example

In this example, each line starts with a symbol indicating what the line represents (e.g., "X" for the reference number, "T" for the title, etc.). The characters following the line symbol represent the musical notation for that line. The "M" line specifies the meter of the piece, the "L" line specifies the length of the notes, the "R" line specifies the rhythm, and the "K" line specifies the key.
ABC notation provides a compact and standardized way of writing down musical pieces that can be easily shared and understood by musicians across different communities and languages.
Among the alternative formats, e.g. Lylipond, ABC notation was chosen as it was the simplest one in terms of representation of the information. As such conversion to the ABC format would be significantly simpler.

3.4.3 Chord construction

Chord construction is the process of generating chords from a root note and a scale. This is a fundamental concept in music theory and it involves using pitches of a scale to create chords that are diatonic to that scale.

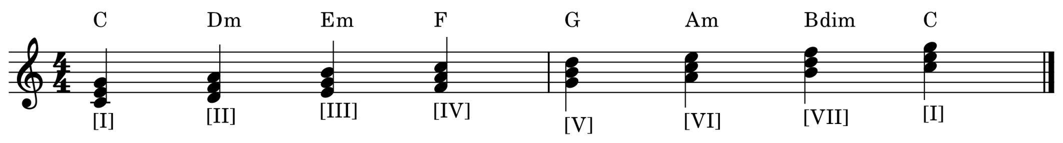


Figure 4: C major chords

I		II		III	IV		V		VI		VII	I
C4	Db	D	Eb	E	F	Gb	G	Ab	A	Bb	B	C5
60	61	62	63	64	65	66	67	68	69	70	71	72

Figure 5: C major degrees / Notes / Pitches

To construct chords from a root note and a scale, the first step is to identify the pitches of the scale that will be used. For example, if the root note is C and the scale is the major scale, the pitches of the scale would be C, D, E, F, G, A, and B.
Next, the chords that can be constructed from these pitches are determined. In the case of the C major scale, there are seven triads that can be constructed, one for each degree of the scale. These triads are C major, D minor, E minor, F major, G major, A minor, and B diminished (Figure 4).
For example, the C major triad consists of the pitches C, E, and G, which are the first, third, and fifth pitches of the C major scale. If we add another one third above the fifth (7th degree), we would get B. This tetrat would be the CMaj7 chord (Figure 5) .
The other triads are constructed in a similar way, using the pitches of the scale that correspond to their root note[7][8].
Of particular interest is the chord at VII degree, the Bdim. This chord has two minor 3rd intervals and as such it contains the tritone[41]. The tritone, which is a 4th augmented interval is unstable and has a strong tendency towards resolution with its tonic.

3.4.4 Chord construction process

The only building blocks we will need for the chord construction process is a root note and a scale. For the scale we will use the intervals which in turn can be turned into actual pitch values. To keep things as simple as possible we will start with middle C, which is also denoted as C4 and has a pitch value of

[41]A minor 3rd, has an interval of 3 semitones. Two minor 3rd's have 6 semitones in sum. The 6 semitones equal 3 tones, which is known as the tritone.

60. We will be using pitch numbers as those are the same as MIDI pitches and it is a standard way of communicating this information. The major scale, also mentioned as ionian mode, will be the main scale for the most of the examples below. The intervals of the ionian mode are the following (Listing 17):

```
(def ionian-intervals [2 2 1 2 2 2 1])
```

Listing 17: Ionian/Major intervals

We can use those intervals to calculate pitches and potentially move things around by changing the root note (Listing 18):

```
(defn intervals->scale [ivs]
  (reduce (fn [acc i] (conj acc (+ i (last acc)))) [0] ivs))

(intervals->scale ionian-intervals)
=> [0 2 4 5 7 9 11 12]

;; we can now easily move to 4th octave by just adding 60
;; to all notes
(mapv #(+ 60 %) (intervals->scale ionian-intervals))
=> [60 62 64 65 67 69 71 72]
```

Listing 18: Major notes calcuation

To construct chords consistently it will be required to move to upper octoves, as notes from the VI degree and onwards will need to borrow notes from the upper octaves.
In the next example, this requirement will be met by using Clojure's lazy and infinite sequences (Listing 19):

```
(def mode-map
  {"major" [0 2 4 5 7 9 11],
   "dorian" [0 2 3 5 7 9 10],
   "phrygian" [0 1 3 5 7 8 10],
   "lydian" [0 2 4 6 7 9 11],
   "mixolydian" [0 2 4 5 7 9 10],
   "minor" [0 2 3 5 7 8 10],
   "locrian" [0 1 3 5 6 8 10]})

(defn intervals* [intervals]
  (mapcat
    (fn [i]
      (map (partial + (* i 12)) intervals)) (iterate inc 0)))

;; if we don't take a specific amount we will get an *infinite*
;; number of notes
(take 20 (intervals* (get mode-map "major")))
```

```
=> (0 2 4 5 7 9 11 12 14 16 17 19 21 23 24 26 28 29 31 33)
```

Listing 19: Major notes calcuation cont.

In this example we provide a key in ABC format and we get a pich, also mentioned as `int-note` in Evolduo (Listing 20):

```
(defn key->int-note [k]
  (let [[n acc] k]
    (if (sharp? k)
      (let [abc-note (if acc (str "^" n) (str n))]
        (note-abc-map abc-note))
      (let [abc-note (if acc (str "_" n) (str n))]
        (note-abc-map-flats abc-note)))))

(key->int-note "C") => 60
```

Listing 20: Key to pitch conversion

Now we can put everything together and construct chords. Chords as explained previously consist of specific intervals. The chord at the first degree (I) of C major scale, would have the notes 60, plus the note one third above that that would be 4 (0 being the 1st and 2 being the second - see major scale intervals above), which would sum into 64, plus one third above that which would be a 3. This way we see that the notes of the chord at the 1st degree of C major scale are 60, 64 and 67 (Listing 21):

```
(def chord-interval-map
  {"R" [0], "R + 5 + R" [0 4 7], "R + 3 + 3" [0 2 4], "R + 3 + 3 + 3" [0 2 4 6]})

(defn gen-chord-notes [{:keys [key mode degree chord]}]
  (let [root-note (key->int-note key)
        scale-notes (intervals* (mode->scale mode))
        chord-notes (map #(+ root-note (nth scale-notes (+ degree %)))
                         (get chord-intervals-map chord [0 2 4]))]
    chord-notes))

(gen-chord-notes {:key "C" :mode "major" :degree 0 :chord "R + 3 + 3"})
=> (60 64 67)
```

Listing 21: Chord notes generation

3.5 Chromosome structure

Chromosomes need to encode two types of information, note pitch and duration. As can be seen in literature[9], in many cases note's pitch representation aligns with the MIDI specification, and duration is an arbitrary number that corresponds to its music equivalent (whole, half, eight etc.). This encoding has

the benefit that it is compact, but the drawback is that genetic operators can easily produce choromosomes which duration's isn't consistent. This can be mitigated with post-processing steps or with smarter genetic operators, but the complexity that would be introduced made those approaches less compelling for this project.
In this work the chromosome structure follows the format of [10], in which pitches are still represented with their corresponding MIDI values, but duration is represented with the amount of prolongation values (-2).
Let's consider the following example (Listing 22):

```
(def c1 [60 -2 -2 -2 -2 -2 -2 -2 60 -2 -2 -2 -2 -2 -2 -2
         62 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2
         64 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2
         65 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2])
```

Listing 22: Chromosome structure

Here, we have 4 measures as each measure has 16 values. The first measure has a C4 (pitch 60), that is followed by 7 prolongation notes (-2). As such, it has a total duration of 1 + 7 = 8, which is a half. It is followed by a note of similar pitch and duration. The 3rd one has a pitch of 62, that has a duration of 16 which is the entire measure, which is a whole note.
The work that describes this format [10], also supports a value of -1 which represents a rest but support for rests was not implemented.
The way this structure is used also has the limitation that a note can be at most of the 16th duration, and 32th or 64ths can't be represented. By growing the measure notes to 32 or 64 this limitation can be alleviated, but in practice this work just uses halfs, quarters and eights.

3.6 Genetic operators

As with most evolutionary algorithm approaches, Evolduo has crossover and mutation operators.

3.6.1 Helpers

With the use of a functional language, creating small functions that do one thing is very common.
In this section one of the most useful ones will be described.
One of those functions just takes a vector of integers, and creates a data structure of easy digestible information. The implementation of this function is the following (Listing 23):

```
(defn calc-note-times [measure]
  (loop [i 0
         notes []]
    (if (>= i (count measure))
      notes
      (recur
        (inc i)
        (let [note (nth measure i)]
          (if (= -2 note)
            (update-in notes [(dec (count notes)) :duration] inc)
            (conj notes {:note note :duration 1 :index i})))))))
```

Listing 23: Calculating note times

In this example, the loop/recur construct that Clojure has is used which prevents stack consumption. This is useful when there is a need for recursive data processing as JVM doesn't have tail call optimization. If we process the Chromosome from Listing 22 we will get (Listing 24):

```
(calc-note-times c1)
=>
[{:note 60, :duration 16, :index 0}
 {:note 62, :duration 16, :index 16}
 {:note 64, :duration 16, :index 32}
 {:note 65, :duration 16, :index 48}]
```

Listing 24: Calculate note times

This result is much easier to understand and pass for processing to other parts of the code.

3.6.2 Crossover

Evolduo is using a single point cut crossover. There is no application specific code apart from what the Chickn library provides. Here is a simple example:

```
(let [pop [{:genes [0 1 2 3] :fitness 1}
           {:genes [4 5 6 7] :fitness 1}
           {:genes [8 9 10 11] :fitness 1}
           {:genes [12 13 14 15] :fitness 1}]
      random-func (constantly 0.5)]
  ((->operator {::type ::cut-crossover
                ::rate 0.5
                ::pointcuts 1
                ::rand-nth rand-nth
                ::random-point chickn.math/rnd-index
                ::random-fun rand}) {:chickn.core/pop-size 4} pop))
=>
[{:genes [8 13 14 15], :fitness 0, :age 0}
 {:genes [12 9 10 11], :fitness 0, :age 0}
 {:genes [4 5 6 7], :fitness 0, :age 0}
 {:genes [8 9 10 11], :fitness 0, :age 0}]
```

Listing 25: Crossover example

In this example we are constructing a cut-crossover operator, with a rate of 50%, using a single cut point and using the stock random functions Clojure is providing. We see see that the 3rd and 4th chromosomes were chosen and were cut at index 1 (zero based).

3.6.3 Mutation

Mutation is more complex. It is comprised of three different functions: Splitting a note, merging two consecutive notes and altering the pitch of the node.
As can be seen in the code example below the selection of the operation is chosen randomly (Listing 26):

```
(defmethod chops/->operator ::music-mutation
           [{:keys [::chops/rate ::chops/random-func] :as cfg}]
  (fn [_ pop]
    (mapv
      (fn [c]
        (let [measures (music/chromo->measures-count (:genes c))]
          (reduce
            (fn [{:keys [genes] :as c} _]
               (if (> rate (random-func))
                 (let [r (rand-int 3)]
                   (condp = r
                     0
                     (assoc c :genes (ops/alter-random-note-pitch genes))
                     1
                     (assoc c :genes (ops/merge-random-note genes))
                     2
                     (assoc c :genes (ops/split-random-note genes))))
                 c)) c (range (* 4 measures))))) pop)))
```

Listing 26: Mutation operators

To increase the chance of getting the genes mutated we are multiplying the number of measures by four. This also ensures that the mutations are proportional to the length of the track and there is a significant chance getting some genetic diversity.
When splitting a note, the index of a random note and the one that follows it are selected. In the middle index of those two notes there is a prolongation gene (-2). The prolongation is altered to a pitch the value of which will be randomly chosen out of 5 values, starting with a tone down, semitone down, same value, or semitone up or tone up. In numeric values that is a random value in [-2, 2] which is added to the pitch of the first note.
The implementation of note splitting is done with the use of a helper function (Listing 27):

```
(defn split-note [c note-idx]
  (assert (not= -2 (c note-idx))
          (str "note a note at idx " note-idx " on " c))
  (let [l (calc-note-length c note-idx)]
    (if (= l 1)
      c
      (let [p  (c note-idx)
            p' (+ -2 (rand-int 5) p)]
        (assoc c (+ note-idx (/ l 2)) p')))))
```

Listing 27: Split note helper function

The actual function is the following (Listing 28):

```
(defn split-random-note [c]
  (let [times (muse/calc-note-times c)
        note (->> times
                  (filter #(note? (:note %)))
                  (filter #(>= (:duration %) 4))
                  shuffle
                  first)]
    (if note
      (split-note c (:index note))
      c)))
```

Listing 28: Split random note

The helper function `calc-note-times` has already been explained in the helpers section (3.6.1).
As randomness is involved (`shuffle` function), the output can vary each time. To reduce verbosity just the first 16 notes are taken in the following example (Listing 29):

```
(take 16 (split-random-note muse/c1))
=> (60 -2 -2 -2 58 -2 -2 -2 60 -2 -2 -2 -2 -2 -2 -2)
```

Listing 29: Splitting random note example

In this example, between the C4 (60) notes at indexes 0 and 8, we are adding a split note at index 4 that has moved one tone down (-2).
Merging notes is the opposite operation of splitting notes. It is finding a note and its next one and the next one is removed by switching to a prolongation. The important bit, as previously, is the fact that we will only merge notes of short durations. For this example we will switch to a different chromosome (Listing 30):

```
(def c2 [62 64 67 -2 -1 -2 -2 -2 67 -2 69 -2 -2 -2 -2 -2])
```

Listing 30: Chromosome #2

Applying merge notes to it can in some cases produce the following (Listing 31):

```
(take 16 (merge-notes muse/c2))
=> (62 -2 67 -2 -1 -2 -2 -2 67 -2 69 -2 -2 -2 -2 -2)
```

Listing 31: Merge notes example

In this case we see the note at index 1 has been switched to a prolongation. It should be noted that this chromosome cannot be produced by Evolduo as the smallest note that can be split is a quarter.
Altering the pitch is a simple function and it will change the pitch of a value 1 or 2 semitones up or down.

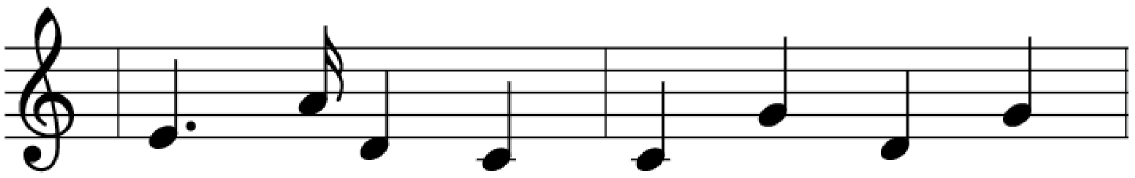


Figure 6: Broken chromosome

3.7 Fixing broken chromosomes

In some cases the genetic operators can produce invalid chromosomes. Invalid chromosomes are those that can't be rendered to ABC because their structure has very short or very long notes. This can result into audible inconsistencies which ideally should be avoided.
Lets consider the following simplified example (Listing 32):

```
(def c3 [64 -2 -2 -2 -2 -2 -2 69 62 -2 -2 -2 60 -2 -2 -2
         60 -2 -2 -2 67 -2 -2 -2 62 -2 -2 -2 67 -2 -2 -2]
```

Listing 32: Broken Chromosome Demo

The staff representation of chromosome c3 is shown in Figure 6.
Someone that has familiarity with reading staff notation will instantly realize that the first measure is not correct. The second note, should be an eight instead of a sixteenth, or there should be a sixteenth rest present. In Evolduo terms, those note times are 6, 1, 4, 4 respectively, which, if summed produce 15 instead of 16.
This is a known problem that isn't visible in practice as mutation will not be allowed to generate 16th notes, and as mutation is happening after crossover this result is also not possible.
A problem the algorithm will fix during a post-processing step is the case where a note will be extended and it will consume a note from the next measure. In the c3 (Listing 32) example if the note at index 12 was selected, this C4 (60) would be extended and the following C4 would be changed to a prolongation. As such, the 2nd measure would start with a prolongation which for the algorithm would mean one of the two possible things, either that the previous note is prolonged to the 2nd measure, or that the 2nd note is transformed into a rest. Neither of those two things are supported at the moment. For simplicity reasons the processing is happening at the measure level and rests are not supported right now.
Let's consider a slightly altered case of c3 (Listing 33):

```
(def c4 [64 -2 -2 -2 -2 -2 -2 69 62 -2 -2 -2 60 -2 -2 -2
         -2 -2 -2 -2 67 -2 -2 -2 62 -2 -2 -2 67 -2 -2 -2]
```

Listing 33: Invalid Measure Times

If we are to to apply the `maybe-fix` function to c4 we would have the following output (Listing 34):

```
(maybe-fix {:key "C"} c4)
=> [64 -2 -2 -2 -2 -2 -2 69 62 -2 -2 -2 60 -2 -2 -2
    60 -2 -2 -2 67 -2 -2 -2 62 -2 -2 -2 67 -2 -2 -2]
```

Listing 34: Fixing Invalid Measure Times

In this example all measures that start with a prolongation will be changed to the root note of the key. We see that the note at index 16 has been changed to C4 (60). This is the simplest possible solution to this problem and there are other ways to tackle this. From using a rest, as mentioned above, to setting it to a value that would consider previous and next notes as well as the mode that track is at so we can use a safe diatonic note. For shorter notes we could just use any in between pitch as a passing note. Those solutions will be considered as future enhancements for this project.

3.8 Fitness Function

In evolutionary algorithms fitness function determines how suitable is a given candidate to survive.
In this problem, fitness determines how good a musical phrase is according to the underlying chord progression. There are many formal musical rules that someone can take into account when evaluating music. The diatonic notes can serve as a good basis to create some music that will sound good to the majority of the listeners. This is usually not enough, and we need to employ more intelligent rules to create something that will have character, something that as Joe Pass says, is worth remembering. The fitness function of Evolduo was inspired by a number of diverse resources, ranging from music theory books to guitar lessons [11, 1, 12, 13]. It should be taken into account that the music rules are following author's music preferences and intuition.
The fitness function in Evolduo is comprised of three different sub-functions:

1. Scale score
2. Last note score
3. Note/chord score

In the following sections the function of those three rules will be documented.

3.8.1 Analyzing chromosomes

While doing a preliminary analysis of chromosomes is not necessarily needed, it can simplify further calculations. The analysis step converts the notes of a chromosome into a data structure with all the necessary data to the follow up tasks. For example, let's assume we have the chromosome from Listing 22.
Performing an analysis will produce the following data structure (Listing 35):

```
(take 2 (analyze {:key "C" :mode "major" :duration 8
                  :progression "I-IV-V-I" :repetitions 1} c1))
=>
({:note 60,
  :duration 8,
  :index 0,
  :chord #{60 64 67},
  :oct-note 0,
  :oct-chord #{0 7 4},
  :type :note,
  :measure-last-note? false}
 {:note 60,
  :duration 8,
  :index 7,
  :chord #{60 64 67},
  :oct-note 0,
  :oct-chord #{0 7 4},
  :type :note,
  :measure-last-note? true})
```

Listing 35: Analyzing a chromosome

For simplicity, only the first 2 elements are taken from the analyzed sequence.
As can be seen the first two notes have the same pitch (60), duration (8, which is 8 sixteenths, meaning a half), type etc. The `oct-note` and `oct-chord` provide the same information as `note` and `chord`, with the different that those are transferred to the 1st octave (after a modulo 12). Reduction to the first octave assists with calculation of notes that have the same pitch in a different octaves. C4 and C5 are considered identical, which is an oversimplification. This will be revisited in the future work of this project.

3.8.2 Scale score

Scale score provides a small score compares to the other two functions explained in the next sections, but it can assist in evaluating chromosomes in minimal setups, for example having a repeating I chord with just the root note, doesn't give much to work with. In such scenarios the other note/chord filter will not be able to give significant measurements.
Scale score will determine how many of the notes fall into the track's scale, multiplying those by 2 and subtracting from the total notes. This small adjustment helps with balancing scores and having a way to provide negative values. For example in the c1 example above (Listing 22) calculating the scale score would give a value of 5 (5 * 2 - 5). If we assume the first note was out of key, e.g. C#, then the score would be 3 (4 * 2 - 5). If all of the notes were out of key we would get -5 and so on.

3.8.3 Measure's last note score

The last note is, in many cases, the most important note. In many styles of music the last note tend to be the root, 3rd or 5th of the key and it is the note the listener will remember. A contrary example would be a 7th which would leave a non-resolved, incomplete feeling.
As described in section 3.8.1, after analysis, the fetching of last notes is just a matter of filtering and calculating the score is just the sum of weighted values, where weight is the duration of the note. The longer the note the more notable is its presence.
While analyzing the last note of each measure can feel incorrect, as some measure's notes may just be continuing in the next one without their last notes being as important as the final notes of a phrase (that can be comprised of multiple measures), it is kept like this for simplicity.

Calculating the last note score is presented below (Listing 36):

```
(def last-note-duration-weight 5)


(defn calc-last-note-score [{:keys [duration oct-note oct-chord type]}]
  (if (and (= type :note)
           (oct-chord oct-note))
    (* duration last-note-duration-weight
       (condp = oct-note
         0 1                                    ;; root
         3 0.7                                  ;; minor 3rd
         4 0.7                                  ;; major 3rd
         7 0.5                                  ;; 5th
         10 -1                                  ;; minor 7th
         11 -1                                  ;; major 7th
         0))
    0)
```

Listing 36: Calculating last note score

Here we see that root, 3rd and 5th are taking a score of 1, 0.7 and 0.5 respectively as 3rds and 5ths are providing an interesting closing option. Sevenths are penaltized with -1 as they produce an unresolved effect. Those values are multiplied by a weight of 5 and, to increase even more the significance of this rule, by the note duration. As a reminder, a whole note will have a value of 16, half 8, etc.
Calculating the overall score becomes as simple as (Listing 37):

```
(defn calc-last-notes-score [analyzed-notes]
  (->> analyzed-notes
       (filter :measure-last-note?)
       (map calc-last-note-score)
       (apply +)))
```

Listing 37: Calcuating all last notes scores

In our example the c1 chromosome will get a last note score of 40, as only the 2nd note (which is the last of the phrase) matches the chord. This note has a duration of 8 which, when multiplied by the weight of 5, produces 40.

3.8.4 Note/Chord score

Note/Chord score rule will penaltize each note that doesn't match the chord. This filter will give a max score of 0 in the case that all notes that are played are chord notes of the underlying progression.
Once again, this value is simple to compute as we have all the necessary data after analysis (Listing 38):

```
(defn calc-chord-note-score [analyzed-notes]
  (reduce (fn [acc {:keys [duration type oct-chord oct-note]}]
```

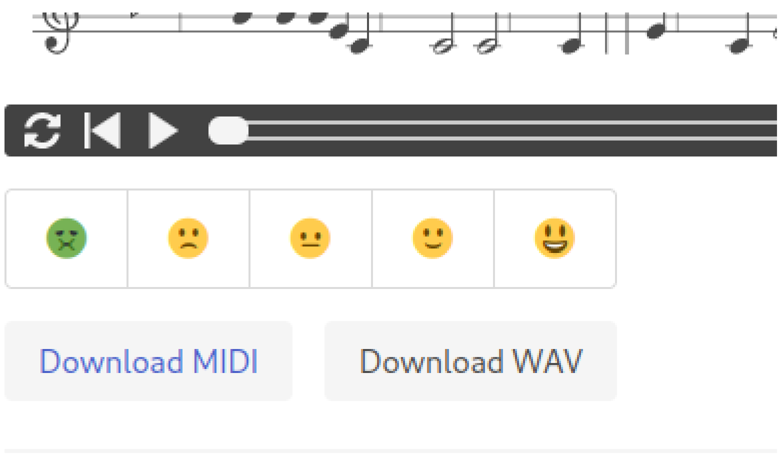


Figure 7: Rating options

```
(if (and (= type :note)
         (not (oct-chord oct-note)))
  (- acc duration)
  acc)) 0 analyzed-notes))
```

Listing 38: Chord notes score

In this example, c1 will get a score of -48 because only the first two notes match the chords, and the next three, each of which has a whole (16) duration will be penaltized, thus, 3 * 16 = 48.

3.8.5 Adjusting fitness score

Evolduo is a platform that also enables users to adjust system's evaluation and potentially perform those actions with the collaboration of other users. This is not a novel idea[14], even though in the majority of other works users don't have those options.
As described previously, fitness is the summed score of three rules that were covered in previous sections (3.8.2, 3.8.3 and 3.8.4).
The fitness produced by the system will be adjusted by the ratings provided by the users.
As can be seen in Figure 7 users have five choices when rating a track. Those choices will produce a value in [-2, 2]. Voting on the middle face has zero effect on the fitness. Voting on the rightmost face doubles the effect comparing to be 4th face etc. As ratings from different users can be provided on the same track, there is a chance, in the case their opinions differ, their ratings will be cancelling each other out.
Initially, there were no calculated stats for each iteration, in which case the summed user rating would be multiplied by 0.2 and the fitness rating. For example, if out of 5 users 4 of them greatly liked the track and one greatly disliked it, we would get a chromosome-rating of 8 (5 * 2 + 1 * -2). If the fitness rating of the chromosome was 50, the adjusted value would be 8 * 0.2 * 50 = 80. The problem with this formula was that when fitness ratings would be close to zero, users' ratings would provide insignificant bias. If the fitness was 0, which is likely as there are parts that produce positive and negative values which when

summed, the adjusted fitness could be a value around zero.
Adjusting this calculation by taking into account min and max fitness values of the entire population of the current iteration is mitigating this issue. In the example above, even if the fitness was zero, the best rating was 100 and the worst 0, we will get a balanced weight (see `*diff` in the code snippet below) of 25. This will push the rating to 200 (8 * 25), thus making this track twice as good as the best track of the current population (in evolutionary rating terms).
In case min and max are around 0, or very low to produce a significant weight, a constant of 10 is used. For example, a min of 0 and a max of 16 will produce a weight of 4, which when below 5 will be ignored. The formula for calculating the adjusted fitness is shown below (Listing 39):

```
(defn calc-adjusted-fitness [ratings stats chromosome]
  (let [chromosome-rating (get ratings (:id chromosome) 0)]
    (math/round
      (+ (:fitness chromosome)
         ;; backwards compatibility
         (if stats
           (let [{:keys [min max]} stats
                 *diff (/ (- max min) 4)]
             (*
               chromosome-rating
               (if (< *diff 5)
                 10
                 *diff)))
           (* 0.2
              chromosome-rating
              (abs (:fitness chromosome))))))))
```

Listing 39: Adjusting fitness rating

3.9 Tackling bias

Bias is an unavoidable side-effect of any work concerning art. There are too many factors that can alter the way users perceive the same information. Those factors can be unrelated to the piece of art itself but still influence user's opinion. Tackling bias is not a trivial problem as it can differ between individuals. Evolduo does a number of things to reduce potential user bias.
When users are rating tracks they don't see the system's fitness rating. People would easily create links between similar tracks, system's evaluations and their scores.
For the same reason, the tracks of an iteration are deterministically shuffled. It's more likely users will listen the tracks in order, from top to bottom. Any score related sorting would potentially introduce a bias of "tracks on the top are better" etc. The shuffling is deterministic to ensure that upon refresh the track ordering remains the same. As a small extra enhancement in this space, shuffling seed is also taking the user id into account so different users are presented with different track orderings. This will help different tracks from the same iteration getting ratings, instead of users rating the same tracks.

User's also don't see other users' interactions on the same track. Users don't know if someone has liked or not a track, and neither if there are any interactions at all. The last one is a questionable choice as giving a hint could also trigger user's interest.

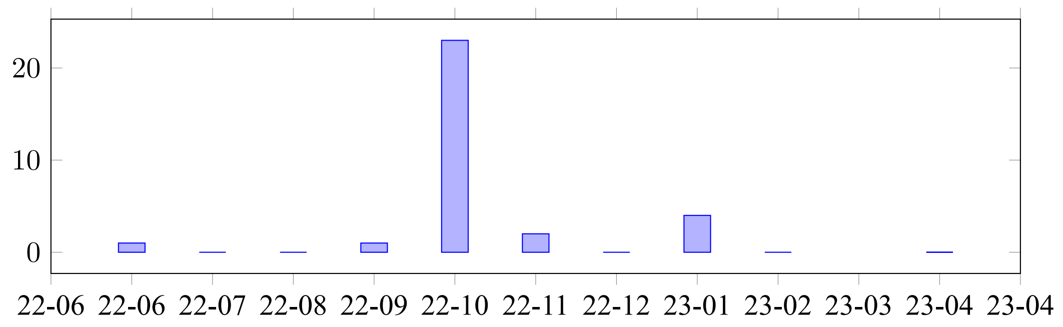


Figure 8: User registrations per month

Chapter 4: Feedback and Results

4.1 Announcement

Evolduo was announced on October 22, 2022 on Clojurians Slack[42] and a few days later (October 28) an announcement was posted on Department of Information and Electronic Engineering of International Hellenic University (IHU) announcement board[43]. Those two announcements brought the majority of users that used the platform. During that time, Evolduo's release announcement was also shared on social networking platforms like Facebook and Twitter, but given the specialized nature of this project it didn't had a significant impact regarding the usage of the platform. Those days usage was monitored at the database level and an extra caution was given to error monitoring.

Evolduo wasn't announced on social networks that had a broader or more specialized scope, such as Reddit, for two reasons. The first had to do with potential errors that could occur on the platform that would be easier to fix without impacting too many users if the usage was limited. The second had to do with reserving the potential announcement for the future, after having collected feedback from the first group of users. It should be noted that the majority of the initial registrations were author's friends who could be available for a direct communication, in case some details were needed. In the next section (4.2), excerpts of discussions will be presented.

After the announcement the usage period was timeboxed to 3 months. Per author's view, this time was considered enough for the platform to be used to some extent. After that, feedback and usage statistics would be analyzed.

At the end of the first 3 month period, Evolduo was also presented on Thessaloniki's Not Only Java meetup group[44]. A complementary announcement was posted on IHU's announcement board[45]. On this event, about 25 people were present and the feedback was positive. This brought a few more registrations. After that the platform wasn't used at all to the time of this writing (Figure 8 and Figure 9).

[42] https://clojurians.slack.com/archives/C06MAR553/p1666422516983289
[43] https://www.iee.ihu.gr/ζήστε-την-μουσική-σύνθεση-χωρίς-απαρα/
[44] https://www.meetup.com/thessaloniki-not-only-java/events/290282680/
[45] https://www.iee.ihu.gr/εκδήλωση-του-thessaloniki-not-only-java-meetup-group-με-θέμα-την-πλατφ/

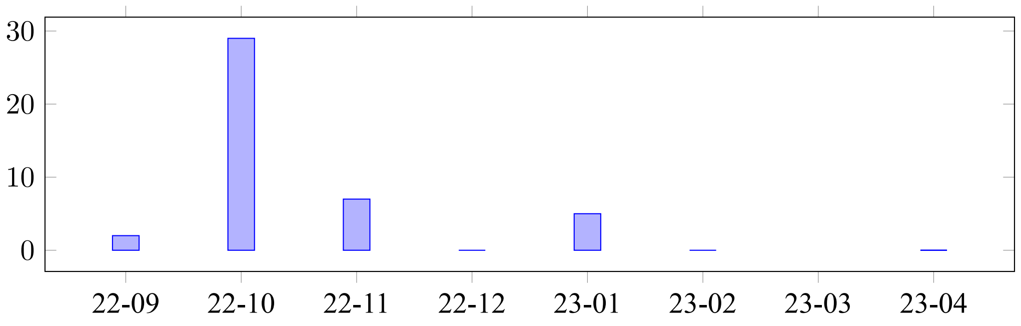


Figure 9: Created evolutions per month

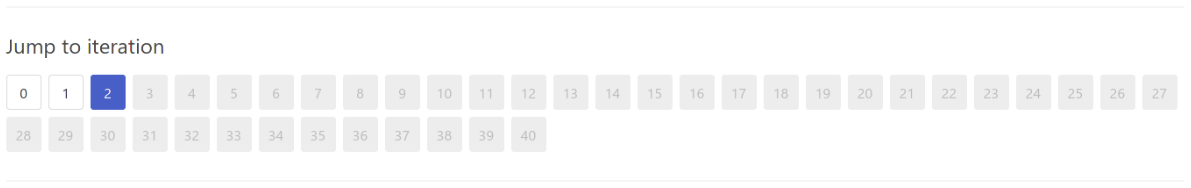


Figure 10: Jump to iteration component

4.2 Feedback

A number of friends provided feedback during the development of this project, during the time of the release and after the project was released.

4.2.1 User Interface

User experience is essential for any non trivial web project and with Evolduo having a very complex set of initial parameters, users reported that it was confusing and off-puting to make an initial selection. Based on this feedback, presets were introduced. Preset is not a unique feature to Evolduo, many complex projects try to simplify configuration by providing a predefined set of initial values.
During the development of the project Kostas K., who specializes in User Interface and User Experience (UI/UX) was providing feedback for the UI of the project. As Evolduo was deployed to a publicly accessible VPS after 6 months since the development was started, the feedback was based on screenshot exchange.
Stathis S. said that the navigation of iterations was the part that confused him the most, it was unclear for him to understand where he was in the evolution process, and that he would prefer having the ability to see the list of past iterations as well as the ones that the process didn't had reach yet. Based on this feedback the "Jump to iteration" UI component rendered all the iterations with the ones not reached yet being displayed as disabled (Figure 10).

4.2.2 Music

Angelos K. was the first person to test the project and provide feedback. Some of his initial reactions and thoughts concerned the ability to have an extended variety of music instruments to choose, larger set of keys - which was restricted to just C at the time because of a chord construction bug, and later,

he suggested that an alternative configuration for the way the rhythm melody was produced would be a good option to have. “The left kind of piano hand that constantly bounces an octave up and down can feel repetitive, it would be nice to provide an alternative where the chord would just be a pad or synth like sound playing for the entire measure”, he said.
An additional idea that Angelos mentioned related to the integration of Evolduo with existing music and audio production tools. “It would be interesting to have Evolduo as a VST, something that we can plug into a Digital Audio Workstation (DAW)”.
During the initial months of the development, Evolduo would generate two separate music voices for a track, one for the melody and the other for the rhythm. After that it was discovered that `ABC.js` could generate the rhythm by just annotating the measure with the chord name. This was a tempting feature to use as it would remove the necessity of each track having two voices, the second of which would not provide much value as it would be the chords of the progression. The flip side to this choice was the fact that the chord construction was implemented (3.4.3), and an option to allow users to select different rhythm patterns would be a useful feature.
Stelios M. provided a big set of ratings and he said that the navigation was the bit that confused him the most. He had a hard time pinpointing the track that he rated before, as he would like to continue listening the next one. This issue was addressed by provided a link to the track that was just rated by the user. He also mentioned that phrygian and locrian modes sounded better to him, especially at high BPM tempos (> 200). This is a good indication that some of the generated tracks can provide a feeling that relates to the musical preferences of the user.
Michael W. opened a github issue[46] and suggested to create a way to have completely random tracks that the user could rate. This could help with the bias problem (3.9) as he perceived all subsequent tracks being better comparing to the previous ones. This wasn’t to suggest that the algorithm did a good job on improving the tracks as much as his “musical expectations where ’preloaded’ in his brain.
Discussion around those points and potential ways to address the aforementioned issues will be presented in the next chapter (5).

4.2.3 Standalone

Another interesting idea people shared, was the potential ability of Evolduo running on a standalone device e.g. a Raspberry PI. As recent Raspberry PIs are capable of running non trivial programs, Evolduo could run on one. This would need some additional work to create a hardware interface to it. Instead of controlling it via a browser, this time, it would need hardware knobs and potentiometers, which would adjust the same controls as one would on the UI. The current form of Evolduo is not suitable for such a task, but potential alternative directions (such as the one presented in section 5.3.7) could work well with this approach. This would enable people using Evolduo similarly as a guitarist would use a guitar pedal effect.

4.2.4 Configuration

Evolduo was presented to a number of people from diverse backgrounds through in-person sessions or video conferences. Almost everyone expressed their confusion regarding the numerous options on the Evolution creation page, as most of the options meant nothing to them.

[46]`https://github.com/kongeor/evolduo-app/issues/1`

Entity	Count
Users	33
Evolutions	44
Iterations	344
Chromosomes	7018
Ratings	249
Invitations	1
Invited users ratings	0

Table 1: Usage statistics

evolution	population	iterations	crossover	mutation	progression	repetitions
#78	20	40	20%	30%	VI-II-V-I	4
#71	40	50	12%	30%	I-I-I-I	2
#70	30	50	5%	90%	I-V-VI-III-IV-I-IV-V	4

Table 2: Sample evolutions

4.3 Results

Evolduo was used mainly from the day of its release announcement on 22th of October 2022 until the end of January of 2023. The number of different entities that were created on the platform can be seen on Table 1. It should come to no surprise that without putting effort into making some fuss on social networking platform or user groups, those sorts of projects don't get much traction. There is a case of a user who probably discovered the platform via the github page, but those are very rare occasions and one can't expect more than a handful of users per year that discover this project by themselves.
On the evolutionary algorithms side, Evolduo is performing as expected. In Figure 11 the best and mean fitness values of an evolution can be observed. For the iteration count, which is not big, as expected there is a constant improvement on both metrics. Evolution #78 has normal genetic configuration without any extremes. Evolution #71 is similar to #78 and its iteration chart can bee seen in Figure 12. What is interesting here, is the fact that even though Evolution #71 has less repetitions the top rated tracks have a similar rating to the #78 (~200-225).
In the case of extreme configuration such as the one in Figure 13 the progress is drastically different. A mutation rate this big (90%) "corrupts" all the best solutions the population has and the tournament selector fails to preserve the best individuals.
In Figure 14 we can see the user rating distribution of the 249 ratings. It's interesting to see that users tend to avoid greatly favoring or not favoring a track and usually picked the in-between choices. This may be an indication of average results.
In Figure 15 we can see how the ratings are distributed progress wise. It should be noted to generate those statistics, we need to compensate for the difference in the number of iterations for each evolution. For an evolution of total 20 iterations, a rating at the 10th falls under the 50% of the progress. On an evolution that has 40, for a rating to fall under the 50% bucket we need to be around the 20th iteration. The corresponding SQL for generating this diagram can be found on Listing 41.
On this diagram it's clearly visible that the majority of users will give up rating after the first iterations.

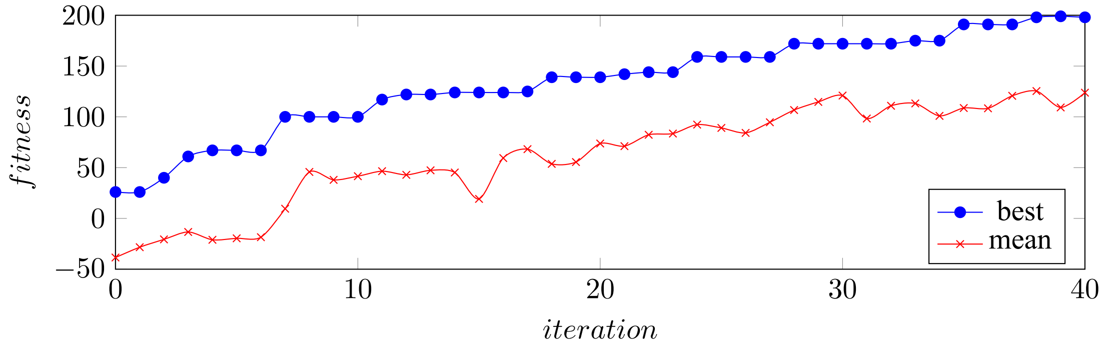


Figure 11: Evolution #78 progress

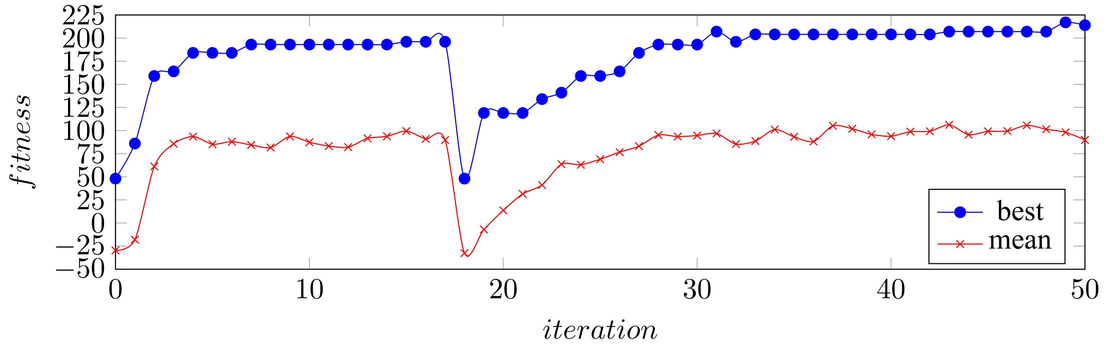


Figure 12: Evolution #71 progress

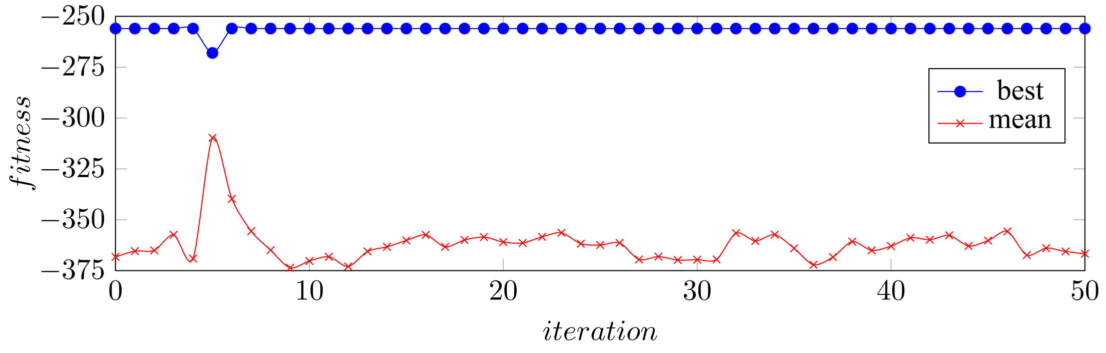


Figure 13: Evolution #70 progress

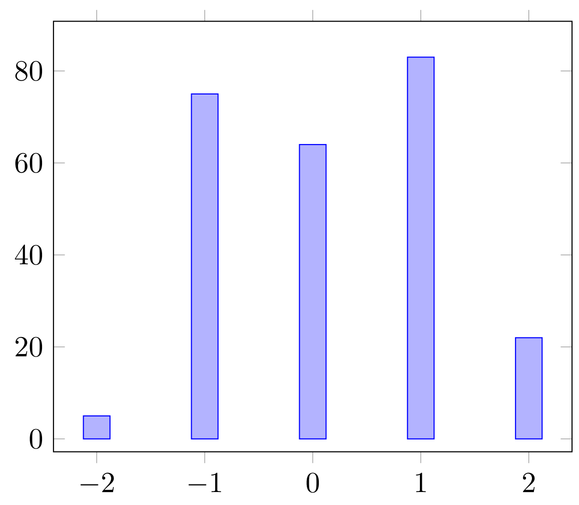


Figure 14: User rating distribution

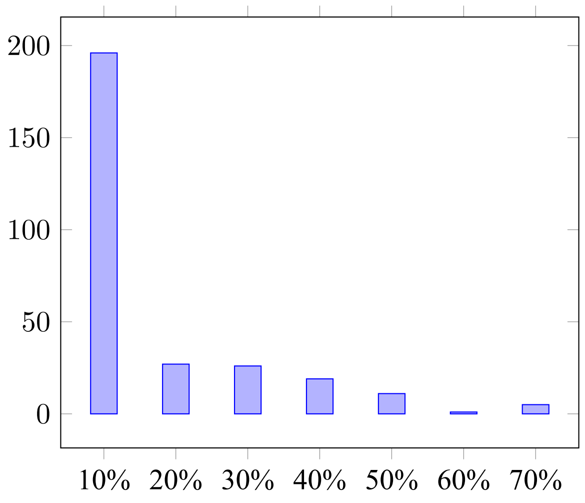


Figure 15: User rating aggregation per evolution progress (balanced)

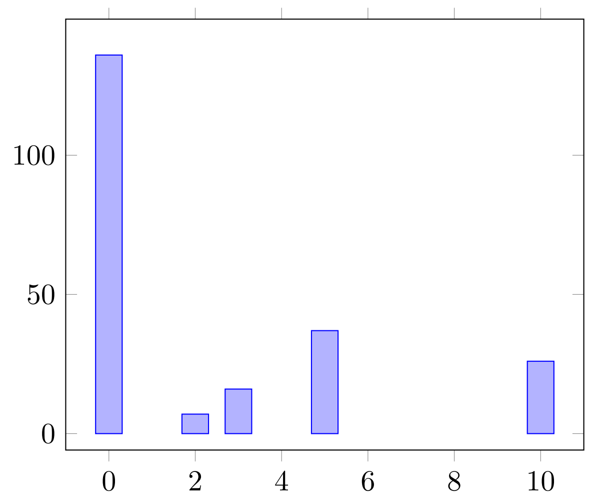


Figure 16: User rating aggregation at first 10% of the process (balanced)

In Figure 16 we see a breakdown of the first 10%, and once again we see that the majority of the ratings are concentrated on the first iteration. This shows that users will give up after the first iteration, something that may be attributed to a number of different reasons that will be explained in the next chapter (5).

Chapter 5: Discussion and Future Work

5.1 Discussion

5.1.1 Evolutionary Algorithms

For music generative tasks evolutionary algorithms provide an interesting tool. The nature of EAs and their ability to explore different solutions for a particular problem, especially a problem where a perfect solution is unfeasible or not required, gives an interesting set of results. Using proper configuration, like tournament selector which avoids choosing the same solutions, thus maintaining genetic diversity, it becomes evident that even after some generations the algorithm doesn't get stuck to a local minimum (e.g. Figure 11).
Crossover allows for combining potentially good solutions that exist in the population. Sometimes there is a feeling that a track starts in a nice way but doesn't end well, or vice versa. Crossover can potentially fix that.
Mutation helps with deviating from the initially musically correct yet monotonous melody towards a more unpredictable and captivating composition. Shifting notes to their adjacent pitches helps exploring the non diatonic areas of the track effectively capturing the user's attention. Despite the occasional occurrence of non-diatonic notes sounding like out-of-key errors, the algorithm remains capable of adjusting them in subsequent generations. Moreover, it can combine them with other not ideal solutions to produce something that is interesting.
The other EA settings, such as population size and random generator functions appear to be sufficiently appropriate for this problem, and they do not appear to require any modifications.

5.1.2 Fitness

The three functions that comprise the fitness calculation seems to enable the exploration of different areas as intended. The results exhibit diversity, and the fitness seems to satisfy in many cases the expectation of what the preset or setup tries to emulate.
Allowing users to influence the fitness score of solutions in each generation provides an alternative means of exploration. It's expected that the algorithm, no matter how diverse, will shape a path based on its fitness rules, but user input can shift that to a different direction.

5.1.3 Presets

Trying to encode rules that don't focus on a particular aspect of the problem is challenging, considering that different music styles and setups often call for distinct approaches. For instance, minimalistic music may favor simple note durations and repetitions but complex jazz forms may require the exact opposite. It's interesting to observe how the current presets strive to meet the described expectations despite using the same set of rules.

5.1.4 ABC and browser interactivity

The ABC format helped with keeping the code as simple as possible, even though, some quirks of the `ABC.js` library resulted in bugs and wrong representations. Despite any issues that appeared the ABC.js was essential for this work as it facilitated a seamless integration into the web application, and enabled users to experience the musical tracks with a simple button press.

5.2 Challenges

5.2.1 Complex configuration

Evolduo is a complex software application, even though it is designed as a web app, it encompasses a substantial amount of preparation and setup to fully experience its functionalities. It goes beyond being a simple application in terms of its intricacy and involved processes. It is targeting a very specific audience, which is primarily musicians. The majority of options are music related aspects, reflecting the primary focus of this work in exploring the musical domain. While the presence of presets attempts to make the application more accessible to users without a music background, the availability of only four presets can be seen as a very limited set of options.
Musicians, who seek to get the most out of this application, may find the extensive EA settings overwhelming, particularly for those without a background in computer science or artificial intelligence.
Given the extensive range of options and combinations, comprehending and exploring all the capabilities of this application would require significant effort and time. The sheer number of meaningful combinations is overwhelming. Unless a user has a very specific interest or, potentially, something to gain from this app, it's unlikely that someone will keep using it.

5.2.2 Time consuming

Evolduo, even in the simplest of cases, produces a massive set of results for a single person to fully process. For instance, a brief composition comprising only four measures at a tempo of 120 beats per minute can yield a track lasting approximately 15 seconds. Listening to just four tracks of an iteration consumes a minute, and when considering 10 iterations, the exploration time escalates to around 10-15 minutes, and that's just to explore a single run of one configuration.
This is one of the reasons people give up very quickly, preventing them from seeing the evolved tracks reaching their full potential, which, in theory, should occur towards the end of the evolution process. As depicted in Figure 11 the evolution keeps progressing but users usually give up before reaching the end.
Another reason for this is that tracks depicted can feel repetitive and, when considering the waiting time in between iterations, it's understandable for people to get easily bored and subsequently give up.

5.2.3 Keeping users engaged

Musicians may find interest, or even practical use, in this app, but for people with no music creation or education background Evolduo is not an easily appealing.
It's not clear what would be a better way to engage users. Typically, users revisit an application with certain expectations, such as the completion of a scheduled task and a desire to observe the resulting outcome. Users may also return to provide feedback on their own creations or collaborations, seeking to witness how things have developed. Additionally, email notifications can serve as prompts, notifying users of updates or progress related to their involvement.
Despite all the possibilities available in Evolduo, it's not clear how to effectively trigger users' interest and make them return to the application.

Bugging constantly users with emails usually has the opposite effect, which is the reason why Evolduo uses email notifications only for the most necessary tasks (email verification, password reset, etc.).

5.2.4 What should an AI music community look like

Evolduo has a feature set that enables building a simple community of users around their interest. People can create tracks, invite others, and co-evolve music, but this fails to generate a sense of engagement. Figures 15 and 16 clearly demonstrate that users give up just right after creating a track or providing their first rating.
One possible explanation for this lack of engagement is that the user interaction is intentionally kept transparent to mitigate potential bias. As a result, users may not perceive themselves as actively participating with others in a collaborative process.
In a system like Evolduo producing a feeling of reward is potentially possible, but it's not clear how to make the social factor engaging to users while maintaining the same properties of the site (AI generated music, exploring a vast set of options etc.).

5.2.5 Supporting a software project

Supporting a software project can be job in itself.
Software requires maintenance, fixing bugs, adding features, addressing security flows, monitoring the announcements of potential CVEs etc. This needs a lot of time.
Additionally, software, and any kind of project, needs to effectively communicate their existence to others. Engaging with people, going to meetup groups or just communicating with people that is also something that requires a lot of time and energy.

5.3 Future Work

5.3.1 EA settings

The initial chromosomes in Evolduo consist of randomly selected notes derived from the underlying chord progression. This design decision aims to provide a starting point that sounds good enough, reducing the number of iterations required to achieve audibly pleasing results. While this is a reasonable choice for this project, it remains uncertain what would happen in alternative initialization cases. Would a completely random initial chromosome be that bad for users? Would it potentially require 10-20 generations to reach a satisfactory outcome?
The initial design of Evolduo took a conservative approach providing satisfactory results with a small number of iterations.
An alternative approach in this context, would be to use actual music phrases from common pop, rock, jazz or even classical music tracks. Would this approach generate more interesting alternatives or would the algorithm distort the original and phrases and generate its own unique variations?
One way to alleviate the trouble of users going through every single iteration, which as seen in the previous chapter leads to user fatigue (Figures 15 and 16), would be to offer users the option to observe the evolution process every 10, 20, 50 or even 100 iterations. This way the increments between what the users observe would be more significant potentially enhancing their engagement and making their experience more efficient, avoiding repetitive or similar outputs. Additionally, this approach would investigate the

convergence points and the limits of the evolutionary process. Currently, convergence is not observed due to the limited number of iterations users can select. It's almost certain that after several hundreds or thousands of iterations the process would converge to some, probably local, optimum.
A single point crossover operator effectively accomplishes recombination, but it could be interesting to explore multi-point crossover as it could produce more diverse results.

5.3.2 Fixing broken chromosomes

In Evolduo, there is a possibility of encountering broken chromosomes, but the extent of their impact and functionality has not been extensively explored. but it is something that hasn't been extensively explored to see how it actually works. The current functionality of the system ensures the process doesn't run into problems that could eventually lead to something that can't be represented in ABC.js. However, it remains unclear how often this phenomenon occurs. This process is also simplistic as it's not needed, or not needed that much, but if more complex setups are explored in the future, it might become necessary to incorporate a more sophisticated approach to deal with broken chromosomes, which would be beneficial for the overall system.

5.3.3 Fitness

The fitness function works well enough to produce interesting outcomes but it relies on the use of arbitrary rules. Based on the outcomes, those rules appear to be reasonable enough but a more precise and systematic approach in this domain would be beneficial.
As mentioned previously the use of known musical phrases as a starting point for the evolution process in Evolduo could be an interesting approach. However, these same tracks could also be employed to evaluate the fitness function's assessment of existing music. This work could be potentially expanded by utilizing A/B tests of actual music phrases with those generated by Evolduo. Users could then participate in a voting process to indicate their preferences. The statistics from these tests could assist in fine-tuning the fitness function based on user preferences which consequently would produce music that would be more interesting for people to listen to.
User's fitness adjustment was also implemented in Evolduo based on author's intuition regarding the impact of user input. There are many open questions in this area that could be explored further. By conducting investigations and studies, the process could be tuned to provide a more effective bias to Evolduo's fitness rating.
Should neutral judgments have any impact on the fitness evaluation? While user input is valuable and should be taken into account, it is also crucial to respect the notion of neutrality. A neutral rating indicates that the user has no specific opinion or preference and this indication should be recognized and considered as a separate category in the evaluation. Ignoring neutral judgments entirely may not be ideal, as they provide valuable information about the user's perspective. Therefore, it is important to incorporate a mechanism that appropriately handles neutral ratings, allowing them to influence the fitness evaluation in a distinct and meaningful manner.
To what extent do the user ratings influence the evaluation process? Does their influence exceed what is considered appropriate? If two users highly rate a track will it displace any other track the system identifies as the best. The limited number of user ratings available makes it difficult to draw definitive conclusions in this domain.

5.3.4 Music

Evolduo offers four presents, that range from minimal to experimental, which serve to initialize the settings and align the music with the desired style. However, to prevent operating as a black box, users are also provided the flexibility to customize these settings according to their preferences. Despite the convenience of this feature, user feedback (4.2.4) indicates that it is still very confusing. There are way too many options and those are very specific to music and EA. As the majority of users do not have the necessary background to understand all the available options, or do not want to explore them, landing on a page with many sliders, drop-down menus and checkboxes can be discouraging.
An alternative approach could be to hide all those options behind an "advanced" link, same as search engines do, and provide a user-friendly wizard that could assist users in setting up their Evolution process based on concepts they understand. For example, users could be asked what kind of feeling should the produced music give. If they choose "happy", the system will select a major or a mixolydian mode. On the other hand, if they select "mellow", aolian mode will be selected, while "dark" could correspond to the phrygian mode, and so on. Some questions could apply to multiple settings, e.g. track length could be affecting the tempo, repetitions and the chord progression. A further potential enhancement that could also collerate to fitness tuning and chromosome initialization from previous sections (5.3.1 and 5.3.3), would be to ask users what they would like the produced music to sound like, based on a genre, an artist or a song. These enhancements are not trivial and could significantly improve the usability of the application.
Currently, there are several music concepts that are not supported within the application. The support for rests is probably the first enhancement in this area that should be addressed. Rests are elementary in all kinds of music and the current chromosome format supports it. What is less trivial is to incorporate rests into the EA operators and it becomes even more challenging to adjust the rules described in section 3.8 to take rests into consideration. As soon as those details are defined, support for rests could be added without requiring radical changes offering users more comprehensive musical composition capabilities.
Currently, Evolduo is limited to supporting only a 4/4 time signature. Expanding its capabilities to include time signatures such as 3/4, 2/4, 7/8 etc. would be an interesting addition. This modification would have implications for the EA processing and fitness evaluation, but it should be feasible to implement without requiring extensive changes. Different time signatures would offer users more diverse rhythmic possibilities.
Support for triplets would also be a nice enhancement, but it is unclear how it should be handled. Most likely with the addition of a different gene code that is similar to the rest gene value (-1). The problem with triplets is the complication of genetic operators as triplets can't be easily represented. For example, a 60 gene followed by 7 prolongations is representing a half note. A half note is the equivalent of three groups of 8th triplets. Addressing this issue would require modifying the rest gene and most likely introducing an additional marker gene that will change the time conversion of notes. At this moment it's not clear what would be an optimal representation for this information.
Evolduo incorporates seven common modes used in western music offering a substantial exploration space when combined with other settings. However, there are several potential extensions that could be implemented to further enhance the system's capabilities. There could be added more modes, especially modes that are found in traditional music like harmonic minor scales, pentatonics, blues scales, or altered scales (e.g. altered phrygian dominant). An option could be added for combining different scales for the

chord construction and the melody generation. For example a major mode could be used for chords with the combination of a pentatonic for the melody, a common setting found in blues and rock music.
In its current state, chord progression is a crucial setting in Evolduo, as well as in music composition in general. The impact and significance of chord progressions in shaping chromosomes are evident within the system. Moreover chord progressions define the length of the chromosome. With the minimum length set at 4 chords and the maximum at 16, it's clear that different settings produce radically different results. For simplicity Evolduo offers a predefined selection of common chord progressions (I-IV-V-I, II-V-I etc.) but as the processing algorithm is generic enough there are no inherent limitations to working with arbitrary, and potentially contrived, progressions.
Currently, each chord in the progression plays for an entire measure. This is very practical from the calculations stand point, but it would be interesting to explore the potential impact of incorporating more complex progressions. For example, one of the most commons I-IV-V-I could be expanded to I - - - / IV - - - / V - - - / I - - - which would allow for adding customizations at the quarter level e.g. I - V - / IV - - - / V - VII - / I - - -, where dash (-) represents the extension of time of the chord. In the previous case, in the 1st measure the I chord would be played for a half (two quarters). This would enable the addition of passing chords that could make the progression much more interesting.
The current chord construction is also quite limited. Presently, the same chord construction applies to all chords within the progression. Support for customizing individual chords could be easily added, for example instead of having all chords being 7ths, the progression could denote which one should be a 7th chord e.g. I - IV - V7 - I. This will require some extension in the parsing mechanism but the functionality is already there. Similarly chord inversions could be added e.g. V7/5 a 7th five chord with the 5th in the bass. Although it would require some modifications to the chord construction process, in general it is a straightforward modification.
Combing the extensions described above and adding a text field for customizing the chord progression could be an interesting enhancement and something that could enable users to further explore the generation of AI driven music. It could also serve as a tool for musicians to have a quick and convenient way of listening and experimenting with different chord progressions.

5.3.5 User experience and collaboration

Many of the enhancements mentioned in the previous sections will require corresponding improvements in the user interface (UI). Currently, the UI of Evolduo is relatively simple, but it may appear overwhelming to those who are new to it. There are tables will lots of data, musical staff lists, and many terms that can intimidate users. Therefore, it would be necessary to refine and simplify the UI of Evolduo, making it more accessible and intuitive for all users.
Modern web applications try to be as simple as possible, especially for end users. Evolduo's UI can be redesigned to support a dual mode functionality. This would include the existing expert mode for users familiar with evolutionary algorithms and music concepts, as well as a new basic mode aimed at individuals with no prior knowledge of EA or music theory. As described already, users can answer to simple questions like what type of mood they have or what kind of music they like and the program can generate music accordingly in real-time. Users can then provide feedback on whether they liked the resulting music or not. This way the application will be more accessible to a wider range of users and it will be easier to collect more data for further improvements.
The requirement for users to sign up can potentially be another barrier, but it is necessary to prevent

potential abuse of the service. An enhancement that could mitigate this concern would be the addition of Single Sign-On (SSO) functionality. As mentioned in section 2.6, the decision to not include SSO was driven by privacy considerations. However, considering that most users are already accustomed to using social networks and their associated authentication systems, privacy concerns may not be a sufficient reason to restrict the use of SSO in this application. By incorporating SSO, users would have the convenience of using their existing social network accounts to sign in, reducing the need for separate account creation and facilitating a smoother user experience.

The introduction of a wizard to initiate an interactive evolution process would not only help users engage with the application but could also be further extended to enable collaboration with others. This will need a restructuring of the data presentation but it would help users to stay for longer durations and utilize their time more efficiently within the application.

Introducing a novel interaction approach where a single track is presented at a time, and transitioning to the next iteration occurs immediately after the user rates the current track, could offer an intriguing user experience that can also be personalized. Ratings should be taken into account not just for balancing the fitness score, but they could also help with changing what is presented to the user. If someone rates three consecutive iteration tracks as “very bad”, it could mean that the current evolution doesn’t match with the user’s preferences.

The presentation of less information, could assist with tackling the bias problem which is one of the most significant issues this project is facing.

5.3.6 Infrastructure and monitoring

The infrastructure of a system is typically adjusted based on current needs. Most of the enhancements discussed below will be necessary as the userbase grows and the system is used constantly.

One important addition, regardless of the user base, is the implementation of Sentry’s background process monitoring. Currently, Sentry is only logs HTTP requests, but it is essential to monitor background tasks such as email sending and especially the evolution process. It’s important to get an insight to errors or performance issues related to these tasks.

Adding one redundant instance when doing a deployment would help of ensure zero down-time and prevent potential disruptions to interactive flows that could be implemented.

As these changes are not critical at the moment, they can be implemented as the project progresses and moves forward.

5.3.7 Evolduo Live

When the idea of Evolduo was conceived and the specifications for this project were determined, there was a basic idea regarding this project’s potential evolution to a live interaction system, something similar to GenJam[47].

Towards the end of this project a small prototype of a different mode was introduced, with the name Evolduo Live[48].

The idea behind this modification was to enable an interactive Evolution process and enable users to modify some settings in real time (e.g. tempo, or mutation rate).

[47] `https://genjam.org/al-biles/genjam/`

[48] `https://www.youtube.com/watch?v=7v1CTX6Cl6s`

Evolduo Live can serve a different purpose, primarily assisting in reducing the feedback loop and minimizing the time required for the Evolution process to complete. It should be noted that the algorithmic part of this modification remains intact and the only required changes are the introduction of a Swing UI and a loop that enables a continuous run of the Evolution process. Intermediate results are placed into a queue and played as a long running track.
In order to achieve this, Evolduo needs to run as a desktop application, requiring the utilization of an external system (PureData[49]) to process the messages generated by Evolduo and transmit them as MIDI events to auxiliary software or hardware devices.
While this change is an interesting alternative mode it looses some of the advantages offered by the web version of Evolduo. The main two drawbacks are the reduced convenience for end users and the lack of preserved data for further analysis. The setup is not trivial at all [50] and it requires additional software to be installed, which is something that the majority of users will not do. Maintaining observability on the data, fitness scores and user ratings is essential for this project to be improved.
The main question that remains is the possibility of combining the best aspects of both worlds. Is it feasible to develop a web-based Live version of Evolduo? Although it would need a number of non trivial modifications this apperas to be possible. After an initial delay users could be presented with a track chosen by the system. As they listen to the track and provide feedback, the system would generate a new track and play it right after the first one. There are many interesting paths in this space, as user feedback can have an immediate impact on the Evolution process. If users find the track boring, dynamic adjustments to EA operators could introduce variation, while if the track seems complex, it could be directed to something simpler. To facilitate this, several UI and processing adjustments would need to be implemented, but it seems possible. It should be noted that MIDI is supported by web browsers[51] so this shouldn't be a barrier for integrating it with other software or hardware instruments.

5.3.8 Different user needs

Any future direction should take consideration the diverse needs of different user groups. Evolduo is a project that caters to both musicians and AI researches. Although it primarily focus on the first category, many EA concepts can be confusing to people with no technical, academic, or AI background. It's unlikely that providing documentation for those concepts will be helpful as this is not a trivial domain within computer science.
Musicians will be more interested in exploring the musical capabilities of this project and will require a more extensive feature set on the music side of things. They will focus on integration with external tools and using it in real-time. The possibility of running Evolduo as a self-contained module, independent of a desktop computer and within a web browser, is an aspect that users may desire. A potential approach to this goal is probably using a Raspberry PI, as this would enable retaining the majority of the codebase and the JVM runtime in its current form. This direction would align more closely with the Evolduo Live concept and could give users with a limited set of options specifically designed for small devices with touch screens.
On the other hand, AI researches will value observability over interactivity. Having a database containing all program outputs would be more useful for someone who needs to perform data analysis and optimiza-

[49]`https://puredata.info/`
[50]`https://github.com/kongeor/evolduo-app/blob/main/doc/live.md`
[51]`https://www.midi.org/17-the-mma/99-web-midi`

tion tasks. Step by step runs will also help users monitor the process more easily. Researchers may also focus on the addition of more parameters on the EA side, such as more genetic operators, ways to bias the different fitness rules, customization options for random generators etc.

5.3.9 Moving forward

Evolduo can potentially satisfy both groups, but as is often the case in software development, it will require additional development time and increased software complexity. Whether this will happen or not, is unknown right now, but it's unlikely to extend this project further without an academic motivation or monetary support. The intention, for now, is to keep the project running and potentially introduce small improvements here and there over time.
The hope is that keeping the web application running will provide something useful to other people working in the same domain. Merely providing a thesis, documentation, or paper without the accompanying project would result in a less than ideal experience.

References

[1] Joe Pass. *Jazz lines*. Alfred Publishing, 2006.

[2] Rich Hickey. A history of clojure. *Proceedings of the ACM on Programming Languages*, 4(HOPL):1–46, 6 2020.

[3] John H. Holland. *Adaptation in Natural and Artificial Systems*. The MIT Press, 1992.

[4] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., USA, 1st edition, 1989.

[5] Kostas Georgiadis. EvolTrio. `https://github.com/kongeor/EvolTrio/`, 2010. [Online; accessed 26-March-2023].

[6] Sean Moriarity. *Genetic Algorithms in Elixir*. The Pragmatic Programmers, 2021.

[7] Nicolas Carter. *Music theory*. Createspace Independent Publishing Platform, North Charleston, SC, March 2018.

[8] Kostas Georgiadis. Music theory resources. `https://blog.cons.gr/posts/2023-03-14-music-theory-resources/`, 2023. [Online; accessed 14-March-2023].

[9] Eduardo Reck Miranda and John Al Biles, editors. *Evolutionary Computer Music*. Springer London, 2007.

[10] Kowalczuk. Evolutionary music composition system with statistically modeled criteria, 2017.

[11] Joseph Alexander. *The complete jazz guitar soloing compilation*. www.fundamental-changes.com, November 2015.

[12] David Hamburger. Take 5: Jazz Blues Soloing - Introduction. `https://truefire.com/jazz-blues-guitar-lessons/take-5-soloing/take-5-jazz-blues-soloing-introduction/v44771`, 2023. [Online; accessed 25-April-2023].

[13] Chien-Hung Liu and Chuan-Kang Ting. Evolutionary composition using music theory and charts. pages 63–70, Singapore, 2013. IEEE.

[14] Alfonso Guarino, Delfina Malandrino, Luca Peppe, Michele Spina, Rocco Zaccagnino, and Nicola Lettieri. A social platform designed for music: Learning and making compositions through collaboration. pages 1004–1009, Shanghai, China, 2019. IEEE.

Appendix A: Useful SQL queries

```
select count(*), r.value  from ratings r
join users u on r.user_id = u.id
where u.id not in (1,2,4,6,7)
group by r.value
order by r.value
```

Listing 40: User rating distribution SQL

```
select g.bucket, count(*) as count from
(select i.num, e.total_iterations, ((i.num::float / e.total_iterations) * 10)::int as bucket
from ratings r
join iterations i on i.id = r.iteration_id
join evolutions e on i.evolution_id = e.id) g
group by g.bucket
order by g.bucket
```

Listing 41: User rating aggregration per evolution progress SQL

```
select g.bucket, count(*) as count from
(select i.num, e.total_iterations,
  ((i.num::float / e.total_iterations) * 100)::int as bucket
from ratings r
join iterations i on i.id = r.iteration_id
join evolutions e on i.evolution_id = e.id) g
where g.bucket <= 10
group by g.bucket
order by g.bucket
```

Listing 42: User rating at first 10% of the process SQL