A musical way of representing cryptographic keys

Cendagorta-Galarza Friend, Daniel

Resum— Motivat pel problema existent amb els mètodes actuals de representació de claus criptogràfiques, que intenten representar un munt de lletres i números sense sentit, s'ha desenvolupat un nou mètode de representació. Prenent com a referència el mètode "The Drunken Bishop", el nostre mètode es basa en la reproducció d'una melodia com a representació de la tonalitat. En lloc de dibuixar el camí del bisbe, hem creat un tauler d'escacs compost per notes musicals, creant una melodia a mesura que el bisbe es mou. Aquesta melodia és determinista, ja que no canvia en funció de l'execució, característica necessària a l'hora de representar claus criptogràfiques. S'han desenvolupat dos mètodes diferents per comparar-los amb els anteriors i avaluar si aquesta nova representació s'adapta al marc que s'estudia. A més, com a exemple d'ús, el mètode seleccionat s'ha implementat al protocol SSH.

Paraules clau—Clau criptogràfica, Representació musical, Bisbe borratxo, SSH, Cercle de quintes, md5, fingerprint, randomart, musicalbeeps, keygen.

Abstract—Motivated by the existing problem with the current cryptographic key representation methods, which try to represent a bunch of meaningless letters and numbers, a new representation method has been developed. Taking as reference "The Drunken Bishop" method, our method is based on the reproduction of a melody as a representation of the key. Instead of drawing the path of the bishop, we have created a chessboard composed of musical notes, creating a melody as the bishop moves. This melody is deterministic, as it doesn't change depending on the execution, a necessary feature when representing cryptographic keys. Two different methods have been developed in order to compare them with the previous methods and evaluate whether this new representation fits within the framework under study. Also, as a use-case example, the selected method has been implemented into the SSH protocol.

Index Terms—Cryptographic key, Musical representation, Drunken Bishop, SSH, Circle of fifths, md5, fingerprint, randomart, musicalbeeps, Keygen.

1 Introduction

SINCE some time ago, people have been interacting with computers in many ways to establish new relationships between both of them. The interface between these two is crucial to facilitating this interaction. This interface can be visual-based or audio-based. In this project, we are developing an audio-based human-computer interface. The interaction between human and computer comes with the computer playing audio and the human recognizing this audio as a manner of recognizing a cryptographic key.

When connecting to a server, public key authentication must be done to ensure that the server is the one the client wants to connect. These cryptographic keys can be represented in many ways. Not only written representations but also visual representations such as the ASCII randomart. These tools allow the user to recognize the cryptographic key easily. As Tyler Cipriani says in [1]: "Public key authentication is confusing, even for "professionals". Part of the confusion is that base64-encoded public keys and private keys are just huge globs of meaningless letters and numbers. Even the hashed fingerprints of these keys are just slightly smaller meaningless globs of letters and numbers. Ensuring that two keys are the same means comparing key hashes— fingerprints. When using the md5 hash algorithm, comparing a key fingerprint means comparing 16, 16-bit numbers (and for the uninitiated that means blankly staring at 32 meaningless letters and numbers). In practice, that usually means not comparing 32 meaningless letters and numbers except when strictly necessary: Security at the expense of usability comes at the expense of security."

So, the main purpose of this project is to create an audible representation to compare it with the previous visual representations. We have based our proposal on "The Drunken Bishop" randomart representation, which draws the path of a bishop moving through a chessboard. Assigning musical notes to the different

- *E-mail de contacte: 1459480@uab.cat*

- *Menció realitzada: Tecnologies de la Informació*
- *Treball tutoritzat per: Carlos Borrego Iglesias (DEIC)*
- *Curs 2021/22*

chess boxes, we created a deterministic, random look alike melody generator, which represents the public key as a musical melody. Also, we will use the SSH protocol as a use-case example, where the implementation of these generators could be useful to facilitate the recognition of a public key.

In this paper we will show the fundamental aspects of the project development, starting with a description of what is intended to do and which is the initial situation. Later, the objectives to be achieved will be appointed, detailing the priority level and the execution thread. The methodology that has been followed for the smooth running of the project will be shown, as the planning that has gone on during this period. There will be shown the main aspects of the development of the project and the findings of the survey. Finally, we will expose the future lines for the project to be improved and the conclusions drawn from the development of the project.

2 State of the art

The audible representation we will be developing is based on another kind of representation, visual this time, called "The Drunken Bishop" [1]. This representation is based on a chessboard, which has 17x9 chess boxes. The bishop will move all around the chessboard while drawing its path. When it passes twice through the same chess box, it will change its character, and the same if it passes 3 times or more, as can be seen from Figure 1.

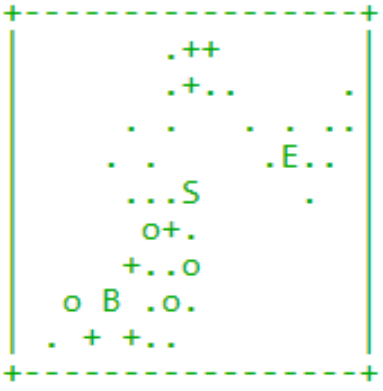

Figure 1: The Drunken Bishop representation.

To determine the movements of the bishop it is used the md5 checksum. The hexadecimal value of the md5 checksum is converted to its binary representation. Later, this will be divided into pairs, representing the bishop's movement. All four combinations of the two bits will be translated into one of the four movements the bishop can do. This way, the algorithm will be deterministic.

To create a musical melody, we have used the Python library "musicalbeeps". This library allows the user to program just in a few lines the reproduction of a single note. It allows to chose the specific note, the duration of the sound, and the relative volume of itself.

3 Objectives

Nowadays the use of SSH is at its peak. The fact that it allows a user to safely connect remotely to a server has made this protocol the most widely used in this area. But, with its increasing popularity, it is being used by more and more users, who are more inexperienced than the previous ones.

The purpose of the project is to program a generator of melodies. This generator of melodies has as a requirement that the melody has to come from an SSH key. It has to be deterministic, therefore, each execution of the program with a given key has to result in the same melody. Also, two programs must be developed to compare them with "The Drunken Bishop" method and determine which method is better for representing SSH keys.

Table 1 is shown the set of objectives to be achieved during the implementation of this project.

Objective	Priority
Theoretical study in search of a technique or tool from music theory that allows creating melodies from a string of zeros and ones.	Essential
Application of music theory to the field of study	Essential
Programming the structure of the chessboard	Essential
Programming the movement of the king through the chessboard	Essential
Programming the creation and reproduction of the melody	Essential
Creation and dissemination of a survey to evaluate the developed methods with "The Drunken Bishop"	Essential
Data processing, statistical study, and conclusions on the developed methods	Essential
Implementation of the chosen method into the SSH protocol	Optional

Table 1: Objectives

These objectives can be classified into these four bigger groups:

- Music theory study
- Programming melodies generator
- Evaluation
- SSH implementation

4 METHODOLOGY AND PLANNING

4.1 Methodology

Due to the nature of the project, it has been decided to use an iterative waterfall methodology. Since each task requires the completion of the previous task and there are groups of tasks that need to be repeated to ensure the proper development of the project, this methodology seemed the right one. It has also allowed studying different approaches to the problem itself, necessary to develop several alternatives.

The project has been developed in a Linux environment, making use of Python, C, and bash, to program the different parts of the project. It has been carried out in a local environment, meant to easily check the correct functioning of the programs.

4.2 Planning

The Planning of the project has been a crucial aspect taking into account the strong dependency between tasks. The uncertainty and lack of knowledge about the topic of study have led to a changing planning over time.

The project has been divided into three main activities. First, to program, at least, two melodies generators, which will form the basis of the project. Second, the drafting of a survey for the later evaluation and comparison of the two implemented melodies generators. And finally, the implementation in the SSH protocol. Finding a good grounding in musical theory, which could be applied to our scheme was a tough job. As well as the drafting of the survey, which, due to the difficulty of comparing visual and auditory methods, was delayed three weeks from what was expected.

In the first instance, it was planned to spend two weeks on the music theory study, which was underestimated. Also, it was planned that were needed four weeks to program the melodies generator. Finally, it took several iterations to find two solid implementations, spending more than two months in total to achieve two final implementations.

5 MELODIES GENERATORS

5.1 Musical theory

The music theory on which the project is based is one of the main pillars and is important to understand. Since fifty thousand years ago, when the first men left Africa, music has been accompanying human beings. But creating it is not that simple. Even for the most talented people in history, composing has been no easy task. A huge amount of studies have been done to try to explain how music works, but none of them has succeeded. However, they have allowed explaining certain aspects of it.

One of the ways to order musical sounds is through the so-called scales [2]. These scales are nothing more than a set of sounds ordered in a certain way, which create a particular sound environment. This set of sounds is not random, they are the resulting notes from choosing a note and multiplying its frequency by 2/1, 3/2, 4/3, etc. This is the way our brain has evolved and that group of sounds all together are what it recognizes as pleasant.

We have used these scales to create the chessboard over which the king is going to move. This way, wherever the king moves, it is going to reproduce a note in the same scale that the ones before. To create these chessboards we have one principle: each note has to be surrounded by all the notes composing the scale. But the scales we chose only have seven notes. So, we add another two composing techniques from music theory, the silence note and the repetition of the previous note. With these two new boxes, we complete all eight boxes surrounding a note and can fill all the chessboard.

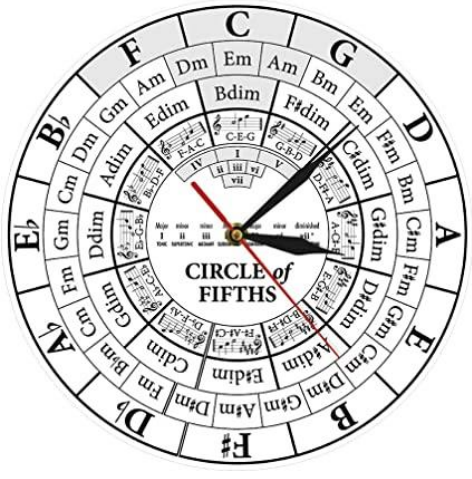

Figure 2: The circle of fifths.

The other music theory concept we have used to

elaborate a melodies generator is the circle of fifths [3]. This theory was created by Nikolay Diletsky in 1679 and is a way of showing the relationships among the twelve semitones of the chromatic scale. In the other melodies generator we spread the notes through the chessboard "randomly", but with this theory, we can elaborate a more correlated chessboard.

This theory doesn't apply directly to the notes inside a scale, but the chords created from those notes. A chord is composed of three notes, which in a major chord will be the note, its third, and its fifth note. We have adapted this circle so we will only have into account the main note of each chord. Figure 2 shows how the circle of fifths is distributed. It is divided into three levels, which are the major, minor and diminished chords levels. This layout allows jumping from chord to chords following some simple rules, to create a pleasant melody. As we are basing our melodies on simple notes, this pleasant effect won't be as noticeable, but still will be a consonant sound, instead of a dissonant one.

To create this chessboard we will simply take the three levels of the circle of fifths and separate them, so the king will be moving through the levels or jumping from one to another.

Another element we are going to count on is time. As said before we have established a box that repeats the note of the previous one. This kind of box allows the melody to create some rhythm, as it breaks with the constant change of notes. Also, to apply more rhythm to the melody is necessary the use of the musical figures, which represent the duration of each note. These musical notes and the associated time can be seen in Figure 3; from left to right, whole note, half note, quarter note, and eighth note.

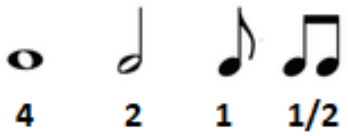

Figure 3: Musical figures.

5.2 King's movement

Now that we have our chessboards created it is time to discuss how the king is going to move through them. First of all, why a king?. As mentioned before, we are basing our representation in "The Drunken bishop" representation, but in our scenario, just four possible movements were too few. The melody would have been way longer. So, we decided that instead of a bishop it would be a king. This way there would be fewer movements and the melody would be Shorter and easier to memorize.

In the case of the scales chessboard the movement is simple, the king will follow the chess rules, being able to move throughout the chessboard, but only one box at a time.

In the case of the circle chessboard, we have based the king's movement on three principles. First, the king can move through the first and second levels with no limit, but not through the third level. This one can only be accessed if coming from the first or second level. And vice versa, from the third level, is only possible to move to the first and second levels. Second, the king can only move through levels to a contiguous box. For example, if the king were in the box "Em" from the shaded area in Figure 2, he could only move to any box inside that shaded area. Third, for the king to have eight possible movements, and to add the silence and repetition boxes, the two remaining movements will be to a silenced box and a repetition box. When this movement is done, the king will return to its previous position to continue with the circle of fifths.

Once the king is moving through the chessboard, the notes of the boxes it moves through will be stored in a list for its later reproduction.

5.3 Key decomposition

An SSH key is a base64-encoded key, which is composed, as Tyler Cipriani says, of a huge glob of meaningless letters and numbers. This key can be hashed, therefore reducing the number of letters and numbers that compose it. In order to generate the melodies, the longer the key is, the longer is the melody obtained. That is why to generate the melody we use the md5 hash algorithm, to obtain a fingerprint of the key and produce with it the melody.

As already mentioned, the melody is created by the movement of the king throughout the chessboard. But, how these movements are determined is what we are going to explain. First of all, we need to transform the hexadecimal md5 fingerprint to binary. Then, the binary string will be divided into chunks of five bits, which will determine each movement. When dividing the string into chunks of five bits, a two bits chunk will be left. These two bits are used to determine, in the case of the scales chessboard, one of the four scales to be used; and in the case of the circle chessboard the note to start with.

From every chunk, we determine the duration of the note, one of the four from Figure 3, and the next movement of the king. The easiest way to implement this was using dictionaries, where the keys are the possible combinations made by the bits and the contents are the duration in milliseconds and the next movement as an increaser or decreaser of the king's actual position.

5.4 Musicalbeeps

Musicalbeeps is a Python package to play sound beeps corresponding to musical notes [4]. This Package is based on the numpy and simpleaudio packages. It allows the user to simply play musical notes, just in a few lines of code. First of all, an instance from the "Player" class must be created, where we will be able to select the desired volume. Then, we will only use one function from this class, "play_note", which plays a beep corresponding to the musical note and the duration that has been passed by parameter. To play a silence, the note "pause" must be played. This library makes the job much easier since it allows us to store the names of the notes in a list and afterward iterate over it to play the notes. This way the melody won't have any interruption or delay based on the execution of the program.

The duration of the note is given by the multiplication of the standard time and the musical figure attached to the note.

6 EVALUATION

To evaluate the developed methods we have decided to elaborate a survey. The elaboration of a survey is a necessary step when developing a project of this nature. In this case, exists the need of comparing the two developed representation methods with the previous one. As our representation is based on "The Drunken Bishop" method, this is the one we are going to compare them with. Also, we find the need of comparing those two methods between them, so that we could decide which is the best one to implement in the SSH protocol.

6.1 Elaboration of the survey

To elaborate on this survey it is important to take into account what are the main characteristics for evaluating the different methods. The purpose of this project is to develop a representation method in order to make it easier to compare the SSH cryptographic keys. So, one of the main characteristics has to be how pleasant the representation in question is. This is a very subjective question since everyone has their own musical tastes. That is why we have decided to define this characteristic as two qüestions. Both are subjective, but somehow, together, they allow to define quite well the issue to be addressed. The first question is direct, which representation seems more pleasant to the interviewee. The interviewees have one example of every representation and they have to decide. The second question follows in the same vein, asking which representation, among three different examples, they perceive as the longest. The duration of the three representations is quite similar, but depending on how pleasant it seems to the interviewee, the shorter it will be felt. This characteristic is also very important to remember the representation. As Jørgen Sugar, a postdoctoral student at the Norwegian University of Science and Technology's Kavli Institute for Systems Neuroscience says in an article published in National Geographic: " Even though time flies when you're having fun, when you remember back on it, you can remember much more of this extended experience compared to a boring experience" [6].

The next two qüestions will allow us to determine, between the most logical standard times, which one is the most appropriate for the circle representation and the scales representation. Here the same melody is played three times, but changing this characteristic.

After having played a few examples of melodies and having shown, also, a few examples of "The Drunken Bishop", we ask the interviewees if they can tell us if they have heard or seen before any of the representation that will be shown. This question is the most objective, since it refers to the main goal of the project, to be able to remember the key representation. The interviewees don't know at the beginning that this question is going to be asked, so they don't pay special attention to every detail, but that is precisely what we want. According to a study carried out by researchers at the University of California in Davis, and published in the Neuron magazine, our brain prioritizes rewarding memòries over other memòries [7].

Finally, since it is normal to access the same server or servers every day, or very often, we ask the interviewees which method do they think they would remember if they hear it or see it every day.

To have a good population sample, we need at least 40 samples and subjects of all ages have been sought. As well as subjects with and without musical theory knowledge, and subjects that play and don't play any instrument. Also, is important to know if the interviewee has an absolute pitch or not, since he/she would have a significant advantage regarding the recognition of the melody. The absolute or perfect pitch is the rare ability of a person to identify or re-create a given musical note without the benefit of a reference tone [8]. This ability is present in a proportion of 1 in 10000 people.

6.2 Results

Taking into account the limits at hand, this survey shows the trend of a group of 40 people. As mentioned before, to be able to analyze the best representation method we have analyzed which option is the most

pleasant for the interviewees.

As can be seen in Figure 4 the circle of fifths seems to be the most pleasant of the method presented. Compared to the other methods this one has a clear advantage since it has more than two-thirds of the samples. If we look at Figure 5 we see the opposite tendency. Here, the circle of fifths has the lowest rate while the scales and the graphic methods increases. This makes sense with the correlation made between those two parameters since it is assumed that the shortest the method seems, the more pleasant it is for the interviewee.

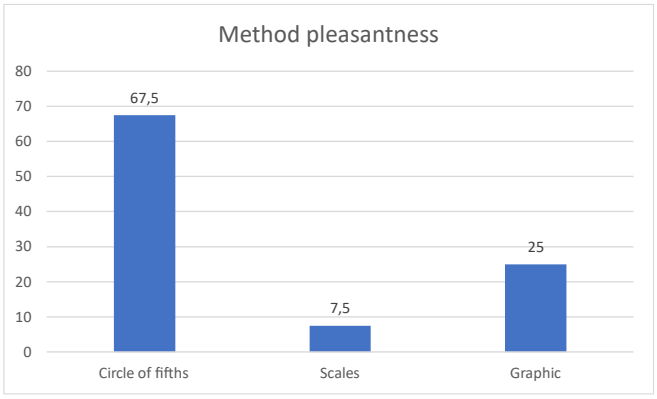

Figure 4: Percentage of method pleasantness

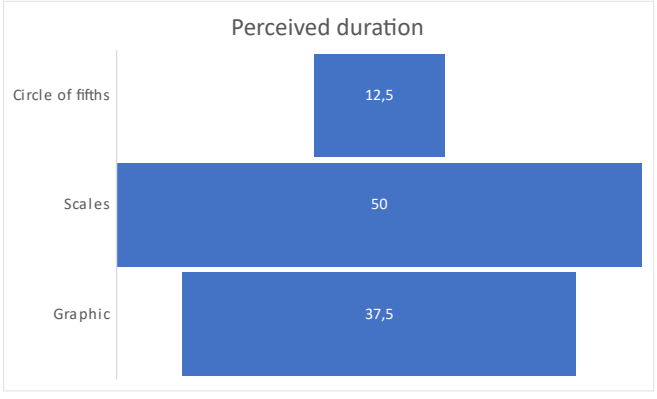

Figure 5: Percentage of longest perceived duration

When deciding which standard time is the best every interviewee agrees between both methods. A 60% of the interviewees find the melody more pleasant when the standard time is 0.15 milliseconds and the other 40% when it is 0.3 milliseconds. None of the interviewees found it pleasant when the standard time is 0.5 milliseconds.

The success rate of every repeated representation can be seen in Figure 6. This rate shows that the circle of fifths is the most correctly guessed when its standard time is 0.15 milliseconds, being followed by the scales with the same standard time and in third position the graphic method. Both scales and circle of fifths methods when played with a standard time of 0.3 or 0.5 milliseconds are the less correctly guessed.

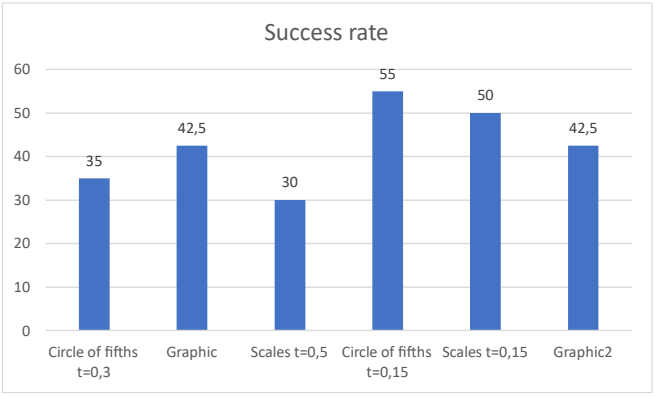

Figure 6: Success rate on representation recognition.

As can be seen in the next images, we found a similar behavior when segregating the samples by musical knowledge and by age. People with musical knowledge seen in Figure 9 and people under 30 years old seen in Figure 7 tend to prefer the circle of fifths method over the other two methods. More than 80% of people with musical knowledge prefer this method and in the case of the people under 30, this percentage decreases to 60%.

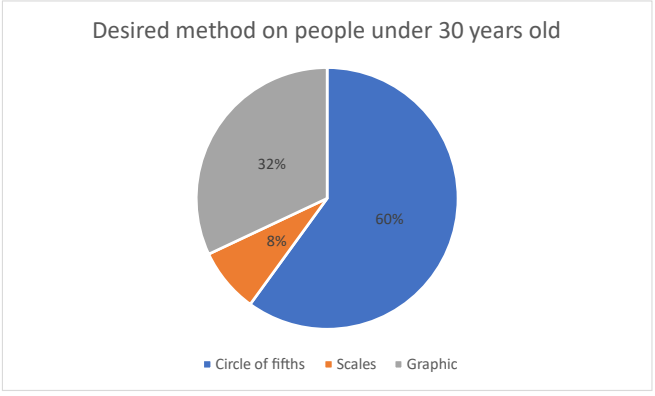

Figure 7: Chosen method people under 30 years old.

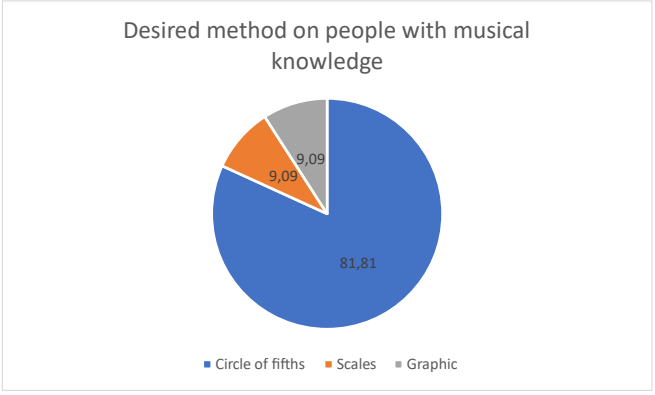

Figure 8: Chosen method people with musical knowledge.

It happens the opposite with people over 30 years old or without musical knowledge as can be seen in Figure 9 and Figure 10. In these two cases, the percentages are pretty similar, more than 60% agree that the graphic method is the best for them.

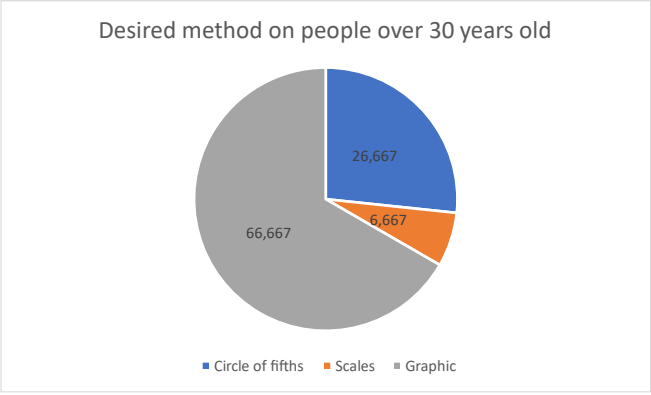

Figure 9: Chosen method people over 30 years old.

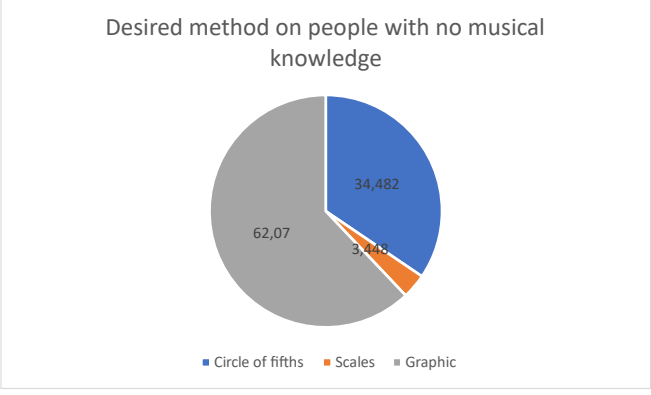

Figure 10: Chosen method people with no musical knowledge.

7 SSH IMPLEMENTATION

To exemplify the use of these generators, it was thought as a good example to implement the chosen one into the SSH protocol. This protocol asks for a cryptographic public key authentication to connect to a host. As mentioned before, this process is not straightforward for everyone since the representation and comparison of these keys is not simple. Among all the existing options to represent these keys, there is none based on the music or any kind of audible representation. That is why this example is important as a first step into what could be the first musical representation of a cryptographic key. We decided to implement it in the OpenSSH portable version, which source code could be found on the OpenSSH website.

The moments at which such representations were to be carried out are mainly two. First, when a new key is generated, the representation must be shown to the administrator. And second, when a public key authentication is requested, where the representation must be shown to the client. The first task of this part of the project was to find the files where these representations were done and include ours. These representations are done in these two files: "ssh-keygen.c" and "ssh-connect.c". Although the behavior of the melodies generator is the same in both examples, the implementation is not, because the first one is shown in the server where the program is running while the second one has to be sent to the client who is trying to connect to the server.

7.1 Ssh-keygen

In this file, the generation of a new public key is carried out in the server. To do this the program asks the administrator for a directory to save the new key, generates it, saves it and finally, the key is shown in both 'base64-encoded' and 'randomart' formats. After these two representations, we will reproduce the melody generated by our melodies generator. As this melody is generated by a Python program, we will need to call it from our OpenSSH code, which is in C. To do this we create a bash shell script, which is called from the C code and executes the Python program with the necessary parameters.

The parameters that the melodies generator needs are the cryptographic key md5 hash and the standard time for the melody. The standard time is a static argument inside the bash shell script, while the cryptographic key has to be sent from the OpenSSH code. In order to do this, the function sshkey_fingerprint has to be called with the "SSH_DIGEST_MD5" option, so it will return the md5 hash of the cryptographic key. As the format in which the fingerprint is returned is different from the one used in the previous executions, minor changes had to be done in the melodies generator Python code.

7.2 Ssh-connect

Unfortunately, this part of the SSH implementation could not be finished due to unsolved problems with the OpenSSH-portable installation. The behavior and functioning are the same that the one from the key generation. When a client wants to connect to a server, it needs to authenticate the server's public key. This key is shown with the 'base64-encoded' and 'randomart' formats and then the melody would be reproduced. The only difference we found within these two parts of the implementation is the communications carried out by the program since in this case, the melody reproduction has to be done by the client, so somehow, the melody has to be sent to the client for its later reproduction.

8 Future lines

Once the project has been finished, there are a couple of improvements that would be interesting to incorporate.

First of all, it would be interesting to improve the circle of fifths generator, implementing the reproduction of the chords instead of the notes. The circle of fifths music theory is based on the relation between chords. In the implemented generator we took the main note of the chord, but not all the notes that compose the chord. This implementation will allow to create more pleasant melodies.

Secondly, regarding the scales generator, it would be also interesting to play the chords corresponding to the scale where the melody is played. It would create an ambient sound that would produce the feeling of belonging of the notes in the scale.

Thirdly, a more complex algorithm could be developed, taking into account diverse aspects of the music theory and bringing them together to create new rules for the ones established.

Finally, it would be important to finish the SSH implementation, adding the reproduction of the melody when connecting to a server. Also, for the melody to be correctly reproduced, it is necessary to run the Python program with no CPU interruptions so that the rhythm of the melody is not affected by the pauses produced by the system.

9 Conclusions

Taking into account the nature of the project, not only a computer science question, but also a study into the different methods to create artificial melodies, the results are very satisfying. Both of the melodies generators have been successfully developed and we have found out that there is an application, where the developed method is one of the better solutions for representing cryptographic keys.

Regarding the planning, all the essential objectives have been accomplished. Even though all the problems found along the way have retarded the execution of the project, we have finished with a very satisfactory result. It is to admit that when the part referring to the generator of melodies was finished, it was thought that the toughest job was done. In the end, the implementation into the SSH protocol took most of the effort and time, even making it impossible to finish this part of the project.

Nevertheless, the music theory study and the application to the presented problem were satisfactorily completed. The melodies generators were also successfully developed and correctly evaluated. And the SSH implementation was partly completed, but still allowed to exemplify the functioning of the project.

Acknowledgments

I would like to thank my tutor Carlos Borrego Iglesias for his unconditional help and for supporting me and encouraging me to overcome the challenges I faced throughout this project. Also, I would like to thank my family and friends who supported me when I was most unmotivated and also for helping me by answering the surveys.

Bibliography

[1] Tyler Cipriani, “Ssh Key Fingerprint, Identicons, And ASCII Art” Sept. 26, 2017. https://tylercipriani.com/blog/2017/09/26/ssh-key-fingerprints-identicons-and-ascii-art/ (accessed Sept. 12, 2021)

[2] Amir Al-Majdalawi Álvarez, “Acustica musical” 2005-2006, https://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_05_06/io2/public_html/escalas.html (accessed Jan. 16, 2022)

[3] Wikipedia, “Circle of fifths” Dec. 28, 2021, https://en.wikipedia.org/wiki/Circle_of_fifths (accessed Nov. 8, 2021)

[4] PyPi, “musicalbeeps 0.2.9” Nov 20, 2020, https://pypi.org/project/musicalbeeps/ (accessed Sept. 23 2021)

[5] Stephanie Briggs, “Surveys 101: A simple guide to asking effective qüestions” Jun. 25, 2015. https://zapier.com/learn/forms-surveys/writing-effective-survey/ (accessed on Nov. 20, 2021)

[6] Dough Johnson, “How you perceive time may depend on your income” Sept 22, 2020. https://www.nationalgeographic.com/science/article/how-you-perceive-time-may-depend-on-income-memory-formation?utm_source=digg (accessed on Dec. 13, 2021)

[7] Mathias J. Gruber et Al, “Post-learning Hippocampal Dynamics Promote Preferential Retention of Rewarding Events” Feb. 11, 2016. https://www.cell.com/neuron/fulltext/S0896-6273(16)00018-0 (accessed on Jan. 2, 2022)

[8] Wikipedia, “Absolute pitch” 13 Jan. 2021. https://en.wikipedia.org/wiki/Absolute_pitch (accessed on Jan. 21, 2022)

APÈNDIX

A1. SOURCE CODE CIRCLE OF FIFTHS GENERATOR

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-

from __future__ import print_function

import argparse
import base64
import hashlib
import sys
import time
import musicalbeeps

player = musicalbeeps.Player(volume=0.3, mu-
te_output = False)

melody = []
notas = []

NOTA = {
    ('0', '0') : 0.5,
    ('0', '1') : 1,
    ('1', '0') : 2,
    ('1', '1') : 4,
}

CASILLA = {
    ('0', '0', '0') : (-1, 1),
    ('0', '0', '1') : (0, 1),
    ('0', '1', '0') : (1, 1),
    ('0', '1', '1') : (1, 0),
    ('1', '0', '0') : (-1, 0),
    ('1', '0', '1') : (0, -1),
    ('1', '1', '0') : (-1, -1),
    ('1', '1', '1') : (1, -1),
}

NOTE_TAB = {
    (0, 0) : ("C4"),
    (0, 1) : ("G4"),
    (0, 2) : ("D4"),
    (0, 3) : ("A4"),
    (0, 4) : ("E4"),
    (0, 5) : ("B4"),
    (0, 6) : ("F#"),
    (0, 7) : ("D4b"),
    (0, 8) : ("A4b"),
    (0, 9) : ("E4b"),
    (0, 10) : ("B4b"),
    (0, 11) : ("F4"),
    (1, 0) : ("A4"),
    (1, 1) : ("E4"),
    (1, 2) : ("B4"),
    (1, 3) : ("F4#"),
    (1, 4) : ("C4#"),
    (1, 5) : ("G4#"),
    (1, 6) : ("E4b"),
    (1, 7) : ("B4b"),
    (1, 8) : ("F4"),
    (1, 9) : ("C4"),
    (1, 10) : ("G4"),
    (1, 11) : ("D4"),
    (2, 0) : ("B4"),
    (2, 1) : ("F4#"),
    (2, 2) : ("C4#"),
    (2, 3) : ("G4#"),
    (2, 4) : ("D4#"),
    (2, 5) : ("A4#"),
    (2, 6) : ("F4"),
    (2, 7) : ("C4"),
    (2, 8) : ("G4"),
    (2, 9) : ("D4"),
    (2, 10) : ("A4"),
    (2, 11) : ("E4"),
}

"""def md5sum(datum):
    m = hashlib.md5()
    m.update(datum)
    return m.hexdigest()"""

def parse_args():
    ap = argparse.ArgumentParser()
    ap.add_argument('-e', '--base', ac-
tion='store')
    ap.add_argument('-t', '--time', ac-
tion='store')
    return ap.parse_args()

def move_king(king, move):
    mov = list(move)
    notas.append(NOTA[mov[0], mov[1]])
    x = int(king[0]) + CASILLA[mov[2],
mov[3], mov[4]][0]
    y = int(king[1]) + CASILLA[mov[2],
mov[3], mov[4]][1]
    if x > 2:
        x = 0
    elif x < 0:
        x = 2
    if y < 0:
        y = 11
    elif y > 11:
        y = 0
    elif x == 0 or x == 1:
        melody.append(NOTE_TAB[x, y])
    elif x == 2:
        if CASILLA[mov[2], mov[3], mov[4]][0]
== 1:
            melody.append("pause")
        elif CASILLA[mov[2], mov[3],
mov[4]][0] == -1:
            melody.append(melody[-1])
        y = king[0]
    return (x, y)


def play_melody():
    for note, duration in zip(melody, notas):
        player.play_note(note, duration*tEst)



if __name__ == '__main__':

    args = parse_args()
    x = args.base.split(":")
    x.pop(0)
    x = ''.join(x)
    print(x)
    md5 = int(x, base=16)
```

```
    pad, rjust, size, kind = '0', '>', 128,
'b'
    bin_key =
str(f'{md5:{pad}{rjust}{size}{kind}}')

    tEst = float(args.time)

    n = 5
    moves = [bin_key[i:i + n] for i in ran-
ge(0, len(bin_key), n)]
    nota_inicial = list(moves.pop())
    melo-
dy.append(NOTE_TAB[int(nota_inicial[0]),int(n
ota_inicial[1])])
    notas.append(NOTA['0', '1'])

    king = nota_inicial
    try:
        for move in moves:
            king = move_king(king, move)
    finally:
        play_melody()
```

A2. SOURCE CODE SCALES GENERATOR

```
#!/usr/bin/env python
# -*- encoding: utf-8 -*-

from __future__ import print_function

import argparse
import base64
import hashlib
import sys
import time
import musicalbeeps

player = musicalbeeps.Player(volume=0.3, mu-
te_output = False)

FLDBASE = 8
FLDSIZE_X = (FLDBASE * 2 + 1)
FLDSIZE_Y = (FLDBASE + 1)

START = (int(FLDSIZE_X / 2), int(FLDSIZE_Y /
2))

melody = []
notas = []

NOTA = {
    ('0', '0') : 0.5,
    ('0', '1') : 1,
    ('1', '0') : 2,
    ('1', '1') : 4,
}

CASILLA = {
    ('0', '0', '0') : (-1, 1),
    ('0', '0', '1') : (0, 1),
    ('0', '1', '0') : (1, 1),
    ('0', '1', '1') : (1, 0),
    ('1', '0', '0') : (-1, 0),
    ('1', '0', '1') : (0, -1),
    ('1', '1', '0') : (-1, -1),
    ('1', '1', '1') : (1, -1),
}

ESCALA = {
    ('0', '0') : {
        (0, 0) : "B4",
        (1, 0) : "R",
        (2, 0) : "pause",
        (3, 0) : "B4",
        (4, 0) : "R",
        (5, 0) : "pause",
        (6, 0) : "B4",
        (7, 0) : "R",
        (8, 0) : "pause",
        (9, 0) : "B4",
        (10, 0) : "R",
        (11, 0) : "pause",
        (12, 0) : "B4",
        (13, 0) : "R",
        (14, 0) : "pause",
        (15, 0) : "B4",
        (16, 0) : "R",
        (0, 1) : "G4",
        (1, 1) : "C4",
        (2, 1) : "E4",
        (3, 1) : "G4",
        (4, 1) : "C4",
        (5, 1) : "pause",
        (6, 1) : "G4",
        (7, 1) : "C4",
        (8, 1) : "pause",
        (9, 1) : "G4",
        (10, 1) : "D4",
        (11, 1) : "E4",
        (12, 1) :"R",
        (13, 1): "C4",
        (14, 1): "pause",
        (15, 1): "G4",
        (16, 1): "D4",
        (0, 2) : "R",
        (1, 2) : "D4",
        (2, 2) : "pause",
        (3, 2) : "R",
        (4, 2) : "D4",
        (5, 2) : "E4",
        (6, 2) : "R",
        (7, 2) : "D4",
        (8, 2) : "E4",
        (9, 2) : "R",
        (10, 2) : "C4",
        (11, 2) : "pause",
        (12, 2) : "G4",
        (13, 2): "D4",
        (14, 2): "E4",
        (15, 2): "R",
        (16, 2): "C4",
        (0, 3) : "R",
        (1, 3) : "B4",
        (2, 3) : "pause",
        (3, 3) : "R",
        (4, 3) : "B4",
        (5, 3) : "pause",
        (6, 3) : "R",
        (7, 3) : "B4",
        (8, 3) : "pause",
        (9, 3) : "R",
        (10, 3) : "B4",
        (11, 3) : "E4",
        (12, 3) : "R",
```

```
        (13, 3): "B4",
        (14, 3): "pause",
        (15, 3): "R",
        (16, 3): "B4",
        (0, 4) : "G4",
        (1, 4) : "C4",
        (2, 4) : "E4",
        (3, 4) : "G4",
        (4, 4) : "C4",
        (5, 4) : "pause",
        (6, 4) : "G4",
        (7, 4) : "C4",
        (8, 4) : "pause",
        (9, 4) : "G4",
        (10, 4) : "D4",
        (11, 4) : "E4",
        (12, 4) : "R",
        (13, 4): "C4",
        (14, 4): "pause",
        (15, 4): "G4",
        (16, 4): "D4",
        (0, 5) : "R",
        (1, 5) : "D4",
        (2, 5) : "pause",
        (3, 5) : "R",
        (4, 5) : "D4",
        (5, 5) : "E4",
        (6, 5) : "R",
        (7, 5) : "D4",
        (8, 5) : "E4",
        (9, 5) : "R",
        (10, 5) : "C4",
        (11, 5) : "pause",
        (12, 5) : "G4",
        (13, 5): "D4",
        (14, 5): "E4",
        (15, 5): "R",
        (16, 5): "C4",
        (0, 6) : "B4",
        (1, 6) : "pause",
        (2, 6) : "R",
        (3, 6) : "B4",
        (4, 6) : "pause",
        (5, 6) : "R",
        (6, 6) : "B4",
        (7, 6) : "pause",
        (8, 6) : "R",
        (9, 6) : "B4",
        (10, 6) : "pause",
        (11, 6) : "R",
        (12, 6) : "B4",
        (13, 6): "pause",
        (14, 6): "R",
        (15, 6): "B4",
        (16, 6): "pause",
        (0, 7) : "G4",
        (1, 7) : "C4",
        (2, 7) : "E4",
        (3, 7) : "G4",
        (4, 7) : "C4",
        (5, 7) : "pause",
        (6, 7) : "G4",
        (7, 7) : "C4",
        (8, 7) : "pause",
        (9, 7) : "G4",
        (10, 7) : "D4",
        (11, 7) : "E4",
        (12, 7) : "R",
        (13, 7): "C4",
        (14, 7): "pause",
        (15, 7): "G4",
        (16, 7): "D4",
        (0, 8) : "R",
        (1, 8) : "D4",
        (2, 8) : "pause",
        (3, 8) : "R",
        (4, 8) : "D4",
        (5, 8) : "E4",
        (6, 8) : "R",
        (7, 8) : "D4",
        (8, 8) : "E4",
        (9, 8) : "R",
        (10, 8) : "C4",
        (11, 8) : "pause",
        (12, 8) : "G4",
        (13, 8): "D4",
        (14, 8): "E4",
        (15, 8): "R",
        (16, 8): "C4",
    },

    ('0', '1') : {
        (0, 0): "D4#",
        (1, 0): "R",
        (2, 0): "pause",
        (3, 0): "D4#",
        (4, 0): "R",
        (5, 0): "pause",
        (6, 0): "D4#",
        (7, 0): "R",
        (8, 0): "pause",
        (9, 0): "D4#",
        (10, 0): "R",
        (11, 0): "pause",
        (12, 0): "D4#",
        (13, 0): "R",
        (14, 0): "pause",
        (15, 0): "D4#",
        (16, 0): "R",
        (0, 1): "B4",
        (1, 1): "E4",
        (2, 1): "G4#",
        (3, 1): "B4",
        (4, 1): "E4",
        (5, 1): "pause",
        (6, 1): "B4",
        (7, 1): "E4",
        (8, 1): "pause",
        (9, 1): "B4",
        (10, 1): "F4#",
        (11, 1): "G4#",
        (12, 1): "R",
        (13, 1): "E4",
        (14, 1): "pause",
        (15, 1): "B4",
        (16, 1): "F4#",
        (0, 2): "R",
        (1, 2): "F4#",
        (2, 2): "pause",
        (3, 2): "R",
        (4, 2): "F4#",
        (5, 2): "G4#",
        (6, 2): "R",
        (7, 2): "F4#",
        (8, 2): "G4#",
        (9, 2): "R",
```

```
        (10, 2): "E4",
        (11, 2): "pause",
        (12, 2): "B4",
        (13, 2): "F4#",
        (14, 2): "G4#",
        (15, 2): "R",
        (16, 2): "E4",
        (0, 3): "R",
        (1, 3): "D4#",
        (2, 3): "pause",
        (3, 3): "R",
        (4, 3): "D4#",
        (5, 3): "pause",
        (6, 3): "R",
        (7, 3): "D4#",
        (8, 3): "pause",
        (9, 3): "R",
        (10, 3): "D4#",
        (11, 3): "G4#",
        (12, 3): "R",
        (13, 3): "D4#",
        (14, 3): "pause",
        (15, 3): "R",
        (16, 3): "D4#",
        (0, 4): "B4",
        (1, 4): "E4",
        (2, 4): "G4#",
        (3, 4): "B4",
        (4, 4): "E4",
        (5, 4): "pause",
        (6, 4): "B4",
        (7, 4): "E4",
        (8, 4): "pause",
        (9, 4): "B4",
        (10, 4): "F4#",
        (11, 4): "G4#",
        (12, 4): "R",
        (13, 4): "E4",
        (14, 4): "pause",
        (15, 4): "B4",
        (16, 4): "F4#",
        (0, 5): "R",
        (1, 5): "F4#",
        (2, 5): "pause",
        (3, 5): "R",
        (4, 5): "F4#",
        (5, 5): "G4#",
        (6, 5): "R",
        (7, 5): "F4#",
        (8, 5): "G4#",
        (9, 5): "R",
        (10, 5): "E4",
        (11, 5): "pause",
        (12, 5): "B4",
        (13, 5): "F4#",
        (14, 5): "G4#",
        (15, 5): "R",
        (16, 5): "E4",
        (0, 6): "D4#",
        (1, 6): "pause",
        (2, 6): "R",
        (3, 6): "D4#",
        (4, 6): "pause",
        (5, 6): "R",
        (6, 6): "D4#",
        (7, 6): "pause",
        (8, 6): "R",
        (9, 6): "D4#",
        (10, 6): "pause",
        (11, 6): "R",
        (12, 6): "D4#",
        (13, 6): "pause",
        (14, 6): "R",
        (15, 6): "D4#",
        (16, 6): "pause",
        (0, 7): "B4",
        (1, 7): "E4",
        (2, 7): "G4#",
        (3, 7): "B4",
        (4, 7): "E4",
        (5, 7): "pause",
        (6, 7): "B4",
        (7, 7): "E4",
        (8, 7): "pause",
        (9, 7): "B4",
        (10, 7): "F4#",
        (11, 7): "G4#",
        (12, 7): "R",
        (13, 7): "E4",
        (14, 7): "pause",
        (15, 7): "B4",
        (16, 7): "F4#",
        (0, 8): "R",
        (1, 8): "F4#",
        (2, 8): "pause",
        (3, 8): "R",
        (4, 8): "F4#",
        (5, 8): "G4#",
        (6, 8): "R",
        (7, 8): "F4#",
        (8, 8): "G4#",
        (9, 8): "R",
        (10, 8): "E4",
        (11, 8): "pause",
        (12, 8): "B4",
        (13, 8): "F4#",
        (14, 8): "G4#",
        (15, 8): "R",
        (16, 8): "E4",
    },

    ('1', '0'): {
        (0, 0): "G4#",
        (1, 0): "R",
        (2, 0): "pause",
        (3, 0): "G4#",
        (4, 0): "R",
        (5, 0): "pause",
        (6, 0): "G4#",
        (7, 0): "R",
        (8, 0): "pause",
        (9, 0): "G4#",
        (10, 0): "R",
        (11, 0): "pause",
        (12, 0): "G4#",
        (13, 0): "R",
        (14, 0): "pause",
        (15, 0): "G4#",
        (16, 0): "R",
        (0, 1): "E4",
        (1, 1): "A4",
        (2, 1): "C4#",
        (3, 1): "E4",
        (4, 1): "A4",
        (5, 1): "pause",
        (6, 1): "E4",
```

```
        (7, 1): "A4",
        (8, 1): "pause",
        (9, 1): "E4",
        (10, 1): "B4",
        (11, 1): "C4#",
        (12, 1): "R",
        (13, 1): "A4",
        (14, 1): "pause",
        (15, 1): "E4",
        (16, 1): "C4#",
        (0, 2): "R",
        (1, 2): "B4",
        (2, 2): "pause",
        (3, 2): "R",
        (4, 2): "B4",
        (5, 2): "C4#",
        (6, 2): "R",
        (7, 2): "B4",
        (8, 2): "C4#",
        (9, 2): "R",
        (10, 2): "A4",
        (11, 2): "pause",
        (12, 2): "E4",
        (13, 2): "B4",
        (14, 2): "C4#",
        (15, 2): "R",
        (16, 2): "A4",
        (0, 3): "R",
        (1, 3): "G4#",
        (2, 3): "pause",
        (3, 3): "R",
        (4, 3): "G4#",
        (5, 3): "pause",
        (6, 3): "R",
        (7, 3): "G4#",
        (8, 3): "pause",
        (9, 3): "R",
        (10, 3): "G4#",
        (11, 3): "C4#",
        (12, 3): "R",
        (13, 3): "G4#",
        (14, 3): "pause",
        (15, 3): "R",
        (16, 3): "G4#",
        (0, 4): "E4",
        (1, 4): "A4",
        (2, 4): "C4#",
        (3, 4): "E4",
        (4, 4): "A4",
        (5, 4): "pause",
        (6, 4): "E4",
        (7, 4): "A4",
        (8, 4): "pause",
        (9, 4): "E4",
        (10, 4): "B4",
        (11, 4): "C4#",
        (12, 4): "R",
        (13, 4): "A4",
        (14, 4): "pause",
        (15, 4): "E4",
        (16, 4): "B4",
        (0, 5): "R",
        (1, 5): "B4",
        (2, 5): "pause",
        (3, 5): "R",
        (4, 5): "B4",
        (5, 5): "C4#",
        (6, 5): "R",
        (7, 5): "B4",
        (8, 5): "C4#",
        (9, 5): "R",
        (10, 5): "A4",
        (11, 5): "pause",
        (12, 5): "E4",
        (13, 5): "B4",
        (14, 5): "C4#",
        (15, 5): "R",
        (16, 5): "A4",
        (0, 6): "G4#",
        (1, 6): "pause",
        (2, 6): "R",
        (3, 6): "G4#",
        (4, 6): "pause",
        (5, 6): "R",
        (6, 6): "G4#",
        (7, 6): "pause",
        (8, 6): "R",
        (9, 6): "G4#",
        (10, 6): "pause",
        (11, 6): "R",
        (12, 6): "G4#",
        (13, 6): "pause",
        (14, 6): "R",
        (15, 6): "G4#",
        (16, 6): "pause",
        (0, 7): "E4",
        (1, 7): "A4",
        (2, 7): "C4#",
        (3, 7): "E4",
        (4, 7): "A4",
        (5, 7): "pause",
        (6, 7): "E4",
        (7, 7): "A4",
        (8, 7): "pause",
        (9, 7): "E4",
        (10, 7): "B4",
        (11, 7): "C4#",
        (12, 7): "R",
        (13, 7): "A4",
        (14, 7): "pause",
        (15, 7): "E4",
        (16, 7): "B4",
        (0, 8): "R",
        (1, 8): "B4",
        (2, 8): "pause",
        (3, 8): "R",
        (4, 8): "B4",
        (5, 8): "C4#",
        (6, 8): "R",
        (7, 8): "B4",
        (8, 8): "C4#",
        (9, 8): "R",
        (10, 8): "A4",
        (11, 8): "pause",
        (12, 8): "E4",
        (13, 8): "B4",
        (14, 8): "C4#",
        (15, 8): "R",
        (16, 8): "A4",
    },

    ('1', '1'): {
        (0, 0): "F4#",
        (1, 0): "R",
        (2, 0): "pause",
        (3, 0): "F4#",
```

```
(4, 0): "R",
(5, 0): "pause",
(6, 0): "F4#",
(7, 0): "R",
(8, 0): "pause",
(9, 0): "F4#",
(10, 0): "R",
(11, 0): "pause",
(12, 0): "F4#",
(13, 0): "R",
(14, 0): "pause",
(15, 0): "F4#",
(16, 0): "R",
(0, 1): "D4",
(1, 1): "G4",
(2, 1): "B4",
(3, 1): "D4",
(4, 1): "G4",
(5, 1): "pause",
(6, 1): "D4",
(7, 1): "G4",
(8, 1): "pause",
(9, 1): "D4",
(10, 1): "A4",
(11, 1): "B4",
(12, 1): "R",
(13, 1): "G4",
(14, 1): "pause",
(15, 1): "D4",
(16, 1): "A4",
(0, 2): "R",
(1, 2): "A4",
(2, 2): "pause",
(3, 2): "R",
(4, 2): "A4",
(5, 2): "B4",
(6, 2): "R",
(7, 2): "A4",
(8, 2): "B4",
(9, 2): "R",
(10, 2): "G4",
(11, 2): "pause",
(12, 2): "D4",
(13, 2): "A4",
(14, 2): "B4",
(15, 2): "R",
(16, 2): "G4",
(0, 3): "R",
(1, 3): "F4#",
(2, 3): "pause",
(3, 3): "R",
(4, 3): "F4#",
(5, 3): "pause",
(6, 3): "R",
(7, 3): "F4#",
(8, 3): "pause",
(9, 3): "R",
(10, 3): "F4#",
(11, 3): "B4",
(12, 3): "R",
(13, 3): "F4#",
(14, 3): "pause",
(15, 3): "R",
(16, 3): "F4#",
(0, 4): "D4",
(1, 4): "G4",
(2, 4): "B4",
(3, 4): "D4",
(4, 4): "G4",
(5, 4): "pause",
(6, 4): "D4",
(7, 4): "G4",
(8, 4): "pause",
(9, 4): "D4",
(10, 4): "A4",
(11, 4): "B4",
(12, 4): "R",
(13, 4): "G4",
(14, 4): "pause",
(15, 4): "D4",
(16, 4): "A4",
(0, 5): "R",
(1, 5): "A4",
(2, 5): "pause",
(3, 5): "R",
(4, 5): "A4",
(5, 5): "B4",
(6, 5): "R",
(7, 5): "A4",
(8, 5): "B4",
(9, 5): "R",
(10, 5): "G4",
(11, 5): "pause",
(12, 5): "D4",
(13, 5): "A4",
(14, 5): "B4",
(15, 5): "R",
(16, 5): "G4",
(0, 6): "F4#",
(1, 6): "pause",
(2, 6): "R",
(3, 6): "F4#",
(4, 6): "pause",
(5, 6): "R",
(6, 6): "F4#",
(7, 6): "pause",
(8, 6): "R",
(9, 6): "F4#",
(10, 6): "pause",
(11, 6): "R",
(12, 6): "F4#",
(13, 6): "pause",
(14, 6): "R",
(15, 6): "F4#",
(16, 6): "pause",
(0, 7): "D4",
(1, 7): "G4",
(2, 7): "B4",
(3, 7): "D4",
(4, 7): "G4",
(5, 7): "pause",
(6, 7): "D4",
(7, 7): "G4",
(8, 7): "pause",
(9, 7): "D4",
(10, 7): "A4",
(11, 7): "B4",
(12, 7): "R",
(13, 7): "G4",
(14, 7): "pause",
(15, 7): "D4",
(16, 7): "A4",
(0, 8): "R",
(1, 8): "A4",
(2, 8): "pause",
(3, 8): "R",
```

```
        (4, 8): "A4",
        (5, 8): "B4",
        (6, 8): "R",
        (7, 8): "A4",
        (8, 8): "B4",
        (9, 8): "R",
        (10, 8): "G4",
        (11, 8): "pause",
        (12, 8): "D4",
        (13, 8): "A4",
        (14, 8): "B4",
        (15, 8): "R",
        (16, 8): "G4",
    },

}

def parse_args():
    ap = argparse.ArgumentParser()
    ap.add_argument('-e', '--base', ac-
tion='store')
    ap.add_argument('-t', '--time', ac-
tion='store')
    return ap.parse_args()

def move_king(king, move):
    mov = list(move)
    nota = NOTA[mov[0], mov[1]]

    x = int(king[0]) + CASILLA[mov[2],
mov[3], mov[4]][0]
    y = int(king[1]) + CASILLA[mov[2],
mov[3], mov[4]][1]

    x = max(x, 0)
    y = max(y, 0)
    x = min(x, FLDSIZE_X - 1)
    y = min(y, FLDSIZE_Y - 1)

    if NOTE_TAB[x, y] != "R":
        melody.append(NOTE_TAB[x, y])
        notas.append(tEst * nota)
    else:
        melody.append(melody[-1])
        notas.append(notas[-1])

    return (x, y)


def play_melody():
    pla-
yer.play_note(NOTE_TAB[int(nota_inicial[0]),
int(nota_inicial[1])], tEst)
    for note, duration in zip(melody, notas):
        player.play_note(note, duration)



if __name__ == '__main__':

    args = parse_args()
    md5 = int(args.base, base=16)
    pad, rjust, size, kind = '0', '>', 128,
'b'

    bin_key =
str(f'{md5:{pad}{rjust}{size}{kind}}')
    tEst = float(args.time)


    n = 5
    moves = [bin_key[i:i + n] for i in ran-
ge(0, len(bin_key), n)]
    nota_inicial = list(moves.pop())

    NOTE_TAB = ESCALA[nota_inicial[0], no-
ta_inicial[1]]

    melody.append(NOTE_TAB[START[0],
START[1]])
    notas.append(tEst)

    king = START
    try:
        for move in moves:
            king = move_king(king, move)
    finally:
        play_melody()
```