**Title of the Paper   
Subtitle of the Paper**

****

**Matura Paper, Kantonsschule Sargans**

Vorname Nachname, Klasse

**Supervisor:**Titel Vorname Name

**Submitted on:**  
6 January 2020

# Preface

In the pursuit of finding a desirable topic for my Matura project, my objective was to combine two of my passions: chess and coding. I have been playing chess for an extended period. My grandfather first introduced me to the game when I was young. He and my uncle played frequently against each other, and by watching them play, I was inspired to start playing as well. I never indulged in chess competitively, but I enjoy playing with my family and friends. Its simple rules and complicated strategies appealed to me. My second interest revolving around my work is coding. Being raised by parents who were programmers, I came across the subject of computer languages often, however I did not have proper coding experience prior to my work. Despite my limited programming knowledge, I found coding appealing, because of its logical way of thinking and creativity. So, I was driven to learn this skill in the result of my work.

With my goal in mind, I came up with the idea of programming different versions of chess by changing the rules of the age-old game. I was inspired by a website called chess.com that has published multiple variations of chess, which I found enjoyable to play. These alternatives ways of playing chess made the game less serious and more diversified. Consequently, I was excited to come up with my own ideas of ways to play chess. By changing the rules accordingly, my goal was to make chess more enjoyable for people that may not appreciate its strategic complexity and pure memorization.

A game of chess pieces

Description automatically generatedI took inspiration from a quote of the famous chess grand master Bobby Fisher, who said: “I hate Chess very much. Because I know what Chess is all about! It’s all about memorization. It’s all about pre-arrangement… Creativity is lower down on the list.” In this statement, Fisher aims to emphasize that a significant aspect of playing chess involves memorization. Because the starting position of chess is always the same, at advanced levels of chess, the initial moves, known as the opening, are preplanned. In this phase experienced players know the optimal response to each move, which they learn prior the game. Fisher heavily criticizes this aspect of the game because it does not involve creative thinking, rather than rote learning. In response Fisher came up with a new way of playing chess. In this variation, the initial row on each side, where the pieces, excluding the pawns, are positioned, are randomly rearranged. With each new game, players encounter a randomly shuffled board that they have not prepared for. In this way players must prioritize strategy and creativity rather than relying solely on pregame knowledge. This game mode is called Fischer Random Chess or Chess960 and it was published in 1996.

Following my work, Fisher inspired me to mitigate the memorization aspect of the game and focus on creativity.

# Table of Content

[Preface II](#_Toc454175648)

[Table of Content III](#_Toc454175649)

[1 Introduction 1](#_Toc454175650)

[1.1 Objectives and Guiding Questions 1](#_Toc454175651)

[1.2 Procedure and Method 1](#_Toc454175652)

[1.3 Structure of the Paper 1](#_Toc454175653)

[2 Chapter 2 2](#_Toc454175654)

[2.1 Subchapter 2 2](#_Toc454175655)

[2.2 Subchapter 2 2](#_Toc454175656)

[3 Chapter 3 3](#_Toc454175657)

[3.1 Subchapter 3 3](#_Toc454175658)

[3.2 Subchapter 3 3](#_Toc454175659)

[4 Chapter 4 4](#_Toc454175660)

[4.1 Subchapter 4 4](#_Toc454175661)

[4.2 Subchapter 4 4](#_Toc454175662)

[5 Summary 5](#_Toc454175663)

[Bibliography IV](#_Toc454175664)

[Table of Figures V](#_Toc454175665)

[Appendix VI](#_Toc454175666)

[A. Appendix 1 VI](#_Toc454175667)

[B. Appendix 2 VII](#_Toc454175668)

[C. Declaration of Authenticity VIII](#_Toc454175669)

# Introduction

## Objectives and Guiding Questions

## Procedure and Method

At the start of my project, I had to determine in which programming language I wanted my project to be written in. Since I had little knowledge about programming, I opted for an easy to understand and beginner friendly computer language. Python is one of the most well-known starting programming languages, making it an ideal choice for my project. Since I had no prior experience with Python, I had to learn the language from scratch. I started my learning journey with a great introduction tutorial I found on YouTube. The author of the video explains the basic functions and variables of Python and reinforces the information with step-by-step projects. I found this guide helpful and engaging, resulting in my improvement in programming.

Next up, I had to choose a chess code, that I could understand well and that can be modified to create new chess variations. I found my way over to GitHub where thousands of programmers are sharing their code with the world. GitHub search features allowed me to look for a chess code that is written in python. After looking through dozens of chess programs, I finally found one to my liking. The one I found did not have major errors and its board was constructed with buttons. I found working with buttons simple, because each button had a distinct identity and when pressed it can call a function.

GitHub allowed me to save my projects in a history data base, which is crucial in programming. When writing any code, it is important to make many backups to jump back to the code if a long-term error or mistake happens.

For my workspace of my program, I chose visual studio code. Visual studio is a well-known source code editor that I have used in school prior to my work. It’s supporting a wide range of programming languages with loads of extensions. During my work however I came across one problem with this workspace, from time to time it marked that I had an indent error in my code, which wasn’t the case. To fix this I had to rearrange my code, for it not to be shown as an error.

The next step was to fully understand the chosen code functions and variables. When I first inspected the code, it appeared very unclear and was difficult to understand. It ranged with a lot of functions and variables that had a lot of code inside of them. For me to fully understand the program I had to go line by line and understand what each variable and function is for. This took a long time, since the code was written by a more experience Python user than me. So, I had to look up a lot of theory regarding Python. Eventually I started understanding the code better and I got ideas how I could modify the code to create my both chess variants.

During my coding of Chaotic Chess, I took use of modules. A module is a separate coding space, that contains functions that you want to include in your initial code.

## Structure of the Paper

# Initial chess code

## Overview

## Variables

Variables are containers for storing data values.

### Buttons

The chess board in the initial code is displayed by an eight-by-eight field of buttons. In the rest of my paper, when taking about the program, I will refer to the squares of the chess board as buttons. Each button is a variable, and the buttons are named after their corresponding name on the chess board. So, when taking about the c7 button, in the code it is named “c7” and it is on row two and column 3. Each button contains a Lambda function. The purpose of a Lambda function is to pass a unique value, depending on the button to a chosen function.

Arrangement of buttons:

A screenshot of a game

Description automatically generated

### Turn

The turn variable can have two values. It can be equal to “W” standing for white or “B”, which stands for black. With the help of this variable the program can decide which players turn it is, which is essential for determining if the chosen move is legitimate or not. If a legal move has been played, the computer has completed the function “check\_input” (check\_input analyzes the move that the user has played and checks if it is legit or not). The “check\_input function has two “if” statements, if the turn is white and if the turn is black. When the program succeeds in completing either of these statements it sets the turn within the correct statement to the opposite color. So, if white has played a legit move, it sets the turn at the and to black. If white fails in playing a right move, the statement has not been completed and the turn doesn’t change.

### Error

A well-written code must handle its errors. If an error occurs, the program shuts down automatically, and the user is unable to interact with it any further. They have to restart the program and are usually left clueless what aroused the problem. To prevent the program from shutting down, the initial chess code introduces a variable called “error”. The program uses this variable to tell the user what the issue in the interaction was. This variable can be equal to four different values.

If it is equal to “1”, the given move was not legitimate. Following instances can occur: King is put in check after move has been played, the chosen piece cannot move to the destination square, or the user did not select an appropriate chess piece.

When the error is set equal to “2”, there is a checkmate. The game ends and the code should no longer execute its commands.

If error is set equal to “3” the king is currently in check. The player that is in check must choose a move that blocks or escapes the check, otherwise the move is invalid.

Finally, when the error is equal to 4 it is checkmate. These values assigned to the error are important to tell the user what the problem is, otherwise, the program would shut down without telling the user what the issue is.

## Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

### Main

The “main” function is responsible for the process of the initial chess code.

The “main” function serves as the core component for the initial chess code. Its purpose is to coordinate the process of the code by calling other functions. The program starts in the “main” function, from where it calls another function. The newly called functions now triggers yet another function, creating a chain like reaction. At the end, the code returns to the “main” function, ending the cycle.

The “main” function serves as a central hub for the entire chess code. Although this function does not contain a lot of code, its purpose is to coordinates the execution of all other functions. Picture it like a chain reaction: The “main” function sets the code in motion by calling other function, and subsequently, that function triggers yet another, creating a continuous sequence. Ultimately, the code returns to the main function, forming a cycle that starts and ends within the main function.

### Checkinput

The Checkinput function looks at the move played and ultimately determines if it is a legit move, by examining the check condition. If the king of either player is put in check, the pieces must behave differently since the priority of the position is to protect the king.

At the start of this function, the checkinput calls the check\_chosen\_move function, which determines if the selected piece can move to its destination field, not accounting for being in check. If this condition is met, the program determines which players made the move, using the previously mentioned turn variable. Afterwards the code goes through the newly created board position and calculates every possible move. It continues by analyzing if the user that played the move put himself into check. If so the error variable is set to one and it disallows the move played. Then it asks the user to play a move that does not put him in check. When the user finds a move that fulfills this condition, the program looks if the player set his opponent into check. After calculating the moves for the new position, the code looks if this check is avoidable. If not, it is checkmate and the game ends.

A chess board with chess pieces

Description automatically generated

### BtnClick

A lambda function serves as a small anonymous function.

When the user choses a button to interact with, the selected button creates a lambda function, that send a unique value to the btnClick and the btnID function. The btnClick function stores the button itself, its color and its text.

## My Notes

Modules:

1. Chaotic
2. Coin
3. …

Variables:

1. Tkstart
2. Tkend
3. test

Functions

## Classes

A Class is like an object constructor, or a "blueprint" for creating objects. Each Object is defined with a set of parameters.

### Figure

The initial chess code contains one Class named “Figure”. The “Figure” Class is responsible for the creation of the chess pieces, by assigning them with a set of parameters.

The “Figure” class in the initial chess has the following parameters:

Figure(name, object\_name, color, start\_position, possible\_moves)

**Name ??**

**Object\_name ??**

**Color** Is a string containing the letter “W” or “B” standing for white and black, followed by a letter according to the chess piece: (“P” for pawn, “N” for knight, “B” for bishop, “R” for rook, “Q” for queen and “K” for king).

**Start\_position** is the button that the piece currently stands on. The start\_position uses other coordinates compared to the buttons. The rows and column of the position range from zero to seven, that is because in computers start counting from zero. So in order to compare a position of a button and of a object, we have to subtract one from the buttons position. This coordinate system is also true for the possible moves.

**Possible\_moves** are all the moves that the chosen piece can make from its current position.

With the usage of this class and its attributes, the program can create all the 32 chess pieces at the beginning. Because the only two pieces that can move at the start are the knights and pawns, the program assigns them with two possible moves each.

A close-up of a quote

Description automatically generated

B1 = Figure('WB', 'B1', 'WP', (6,0), [(5,0),(4,0)])

B2 = Figure('WB', 'B2', 'WP', (6,1), [(5,1),(4,1)])

B3 = Figure('WB', 'B3', 'WP', (6,2), [(5,2),(4,2)])

B4 = Figure('WB', 'B4', 'WP', (6,3), [(5,3),(4,3)])

B5 = Figure('WB', 'B5', 'WP', (6,4), [(5,4),(4,4)])

B6 = Figure('WB', 'B6', 'WP', (6,5), [(5,5),(4,5)])

B7 = Figure('WB', 'B7', 'WP', (6,6), [(5,6),(4,6)])

B8 = Figure('WB', 'B8', 'WP', (6,7), [(5,7),(4,7)])

T1 = Figure('WT', 'T1', 'WR', (7,0), [])

T2 = Figure('WT', 'T2', 'WR', (7,7), [])

S1 = Figure('WS', 'S1', 'WN', (7,1), [(5,0),(5,2)])

S2 = Figure('WS', 'S2', 'WN', (7,6), [(5,5),(5,7)])

L1 = Figure('WL', 'L1', 'WB', (7,2), [])

L2 = Figure('WL', 'L2', 'WB', (7,5), [])

D1 = Figure('WD', 'D1', 'WQ', (7,3), [])

K1 = Figure('WK', 'K1', 'WK', (7,4), [])

B9 = Figure('BB', 'B9', 'BP', (1,0), [(2,0),(3,0)])

B10 = Figure('BB', 'B10', 'BP', (1,1), [(2,1),(3,1)])

B11 = Figure('BB', 'B11', 'BP', (1,2), [(2,2),(3,2)])

B12 = Figure('BB', 'B12', 'BP', (1,3), [(2,3),(3,3)])

B13 = Figure('BB', 'B13', 'BP', (1,4), [(2,4),(3,4)])

B14 = Figure('BB', 'B14', 'BP', (1,5), [(2,5),(3,5)])

B15 = Figure('BB', 'B15', 'BP', (1,6), [(2,6),(3,6)])

B16 = Figure('BB', 'B16', 'BP', (1,7), [(2,7),(3,7)])

T3 = Figure('BT', 'T3', 'BR', (0,0), [])

T4 = Figure('BT', 'T4', 'BR', (0,7), [])

S3 = Figure('BS', 'S3', 'BN', (0,1), [(2,0),(2,2)])

S4 = Figure('BS', 'S4', 'BN', (0,6), [(2,5),(2,7)])

L3 = Figure('BL', 'L3', 'BB', (0,2), [])

L4 = Figure('BL', 'L4', 'BB', (0,5), [])

D2 = Figure('BD', 'D2', 'BQ', (0,3), [])

K2 = Figure('BK', 'K2', 'BK', (0,4), [])

When looking at the start\_position and possible moves values, we can see number ranging from zero to seven. That is because the pieces use a different grid system when compared to the buttons. If we want to have the same positions for buttons and pieces, it is required to subtract one from the buttons row and column. The a8 square of the chess board is located on the top left corner. Its buttons position is row one and column one. If a piece were standing on that button however, the piece would have the location row zero and column zero. This is due to the fact that computers start counting from zero.

# Color Chess

## Rules of Color Chess

In Color Chess, players are assigned an individual color. When their chess pieces move to a square, that square is highlighted with the player's distinctive color. The goal of the game is to color in more squares with your own color than your opponent does with theirs. Only squares on which the pieces stand on or have been standing on are colored in. If a piece captures the opponent’s piece, it will overtake its color. Since each game of chess starts with the same position, the first two rows on each side will be colored in at the beginning. The game ends after 30 moves. The player who has colored in more squares during this period wins. Alternatively, victory can be attained by checkmating the opponent within these 30 moves.

Starting position:

A screenshot of a game

Description automatically generated

Example of capturing a piece:

A screenshot of a game

Description automatically generatedA screenshot of a game

Description automatically generated

## Coding of Color Chess

### Variables

## At the start of the code the program asks the user to choose a color that represents the white player and black player. This color gets stored inside of a variable and it can be used later to color in the fields occupied by either white or black.

Afterwards two other variables are created, that are responsible for the numbers of squared colored in for white and black. These variables are both set to 16 at the start, since both players have 16 pieces occupying 16 squares.

The last variable that is introduced at the start of the code is how many moves does it take until the game ends. In order to create a balanced game, I opted to set this value to 20. If the rounds played so far are bigger then 20, the game ends.

The count\_turn variable is responsible for determining which player turn it is. After every legitimate move the count\_turn goes up by one. At the start

### Labels

In python a variable stores data and labels are stored with variables aswell.

To make color chess more beginner friendly, I introduced three different labels to the GUI that tell the user what is happening on the board.

Rounds\_label: The first label is responsible for counting how many moves have been played so far. This way the user can keep track of how long the game will last. The rounds\_label requires a variable called “rounds”. This variable stores the number of rounds that have been played so far. To showcase the “rounds” variable to the “rounds\_label” .config() Is used. With .config() every setting of a label can be changed. Since we want to display text with the rounds\_label we can edit its text with rounds\_label.config(text = rounds). However, the following problem would occur. If we want to edit text, we can only display a string. A string in python is something that is surrounded by quotation. Because our rounds variable is simply a number and not a string, we must convert the number to a string, in the following way:

Count\_label .config(text = str(rounds))

Turn\_label: The turn\_label functions in a similar way than the rounds\_label. It stores a variable called count\_turn.

### The coloring of the squares

With the help of the end\_button, which is the button the user clicks on the second time. Subsequently I could target the button that the user moves to. With the before used .config() feature I can change the color of the relevant button. To know in which color to dye in the button, we have to know which player made the move. With the earlier used count\_turn variable I can see how many rounds have been played so far and since there are exactly two player player a move after each other, every second move make the same player. Knowing this we can divide the count\_turn variable by 2, if the solution is a whole number, white made the move. If the solution has a rest, black made the move. Having done that, we can tell our program if the relevant move is made by white it should use white coloring, and so for black.

# Chaotic Chess

## Rules of Chaotic Chess

Chaotic Chess introduces four items that the chess pieces can pick up, granting them special abilities. To make the game balanced, the items are equally distributed on empty fields on the white side (rows one to four) and on the black side (rows five to eight). A piece can pick up an item by stepping on its according square. The items change their positions every four moves successively. The game comes to an end if either king is in checkmate.

The four items consist of a bomb, shield, coin and barrier.

Example of items arrangement:

A screenshot of a game

Description automatically generated

### Bomb

Picking up the bomb triggers an explosion in a three-by-three area. Pieces within that area are destroyed and removed from the board. The piece that initially stepped on the bomb is eliminated as well. If a bomb is near the edges or corners of the board, the explosion radius only takes up as much space as it is granted.

Example of bomb usage:

A screenshot of a game

Description automatically generatedA screenshot of a game

Description automatically generated

Note: In this example white captures the bomb in the middle to take down blacks bishop and pawn, while sacrificing two pawns of their own.

### Coin:

The chess piece that steps on the coin is permanently transformed into the piece represented by the coin. To determine which piece is portraited by the coin, a random choice is made between knight, bishop, and rook. The king is the only piece that is not allowed to pick up the coin.

Example of coin usage:

A screenshot of a game

Description automatically generatedA screenshot of a game

Description automatically generated

Note: This example shows how white uses the coin to turn their pawn into a more stronger piece, the bishop, to give themselves a tactical advantage over the current position.

### Shield:

The shield makes the piece that steps on it invincible until the shield changes its position. Anchored to its square, the shield does not move with the piece that picked it up.

Example of shield usage:

A screenshot of a game

Description automatically generatedA screenshot of a game

Description automatically generated

Note: In this example white is taking advantage of the shield, by moving their queen to a square that is defended by black’s knight. Since the queen is protected by the shield, black cannot capture whites queen with their knight.

### Barrier:

The square that the barrier is placed on, prevents pieces to step onto that field. Only the field that the barrier is placed on is affected, pieces can jump over the barrier.

Example of barrier usage:

A screenshot of a game

Description automatically generatedA screenshot of a game

Description automatically generated

Note: In this example white attacks, the queen with their bishop and since black cannot defend their queen or move their queen to safety, they are forced to give up their queen.

## Coding of Chaotic Chess

# Summary

# Bibliography

Bonati, P., & Hadorn, R. (2009). *Matura- und andere selbständige Arbeiten betreuen* (2. Auflage ed.). Bern: Heb Verlag AG.

Metzger, P. (2010). *Abschlussarbeiten.* Aarau: Sauerländer.

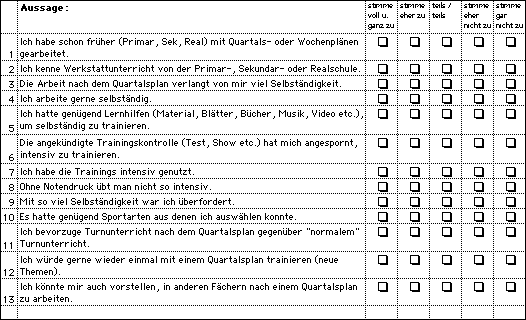
*Panorama*. (2015, December 15). Retrieved December 6, 2019, from Statistik Schweiz - Panorama: http://www.bfs.admin.ch/bfs/portal/de/index/themen/01/01/pan.html

# Table of Figures

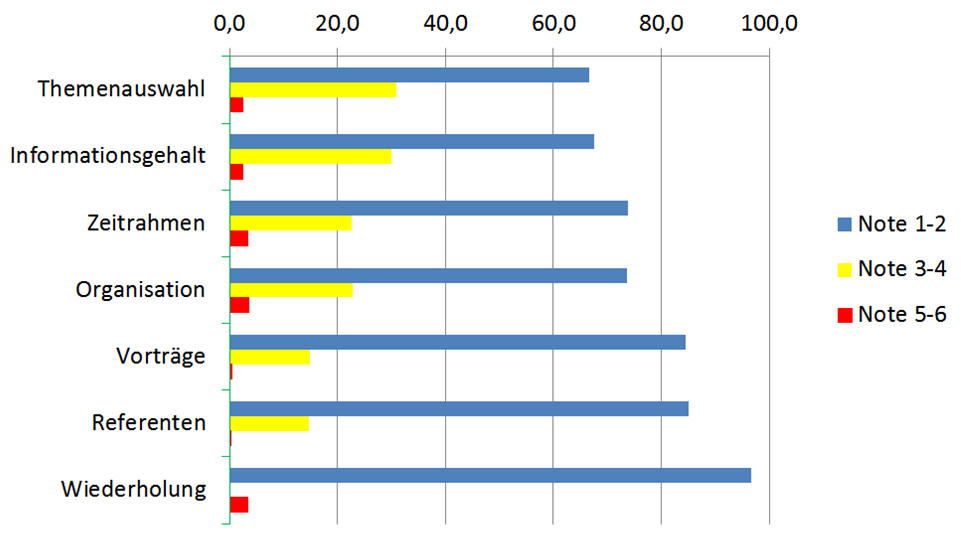
[Figure 1: Demographics of Switzerland in 1900 and 2014 3](file:///D:/Sargans%202016-2017/3-Info/9)%20Maturavorlage%20erstellen/Kurs%20MA-Vorlage%20Reuteler/Vorlage%20Maturaarbeit%20Englisch.docx#_Toc454176099)

# Appendix

## Appendix 1

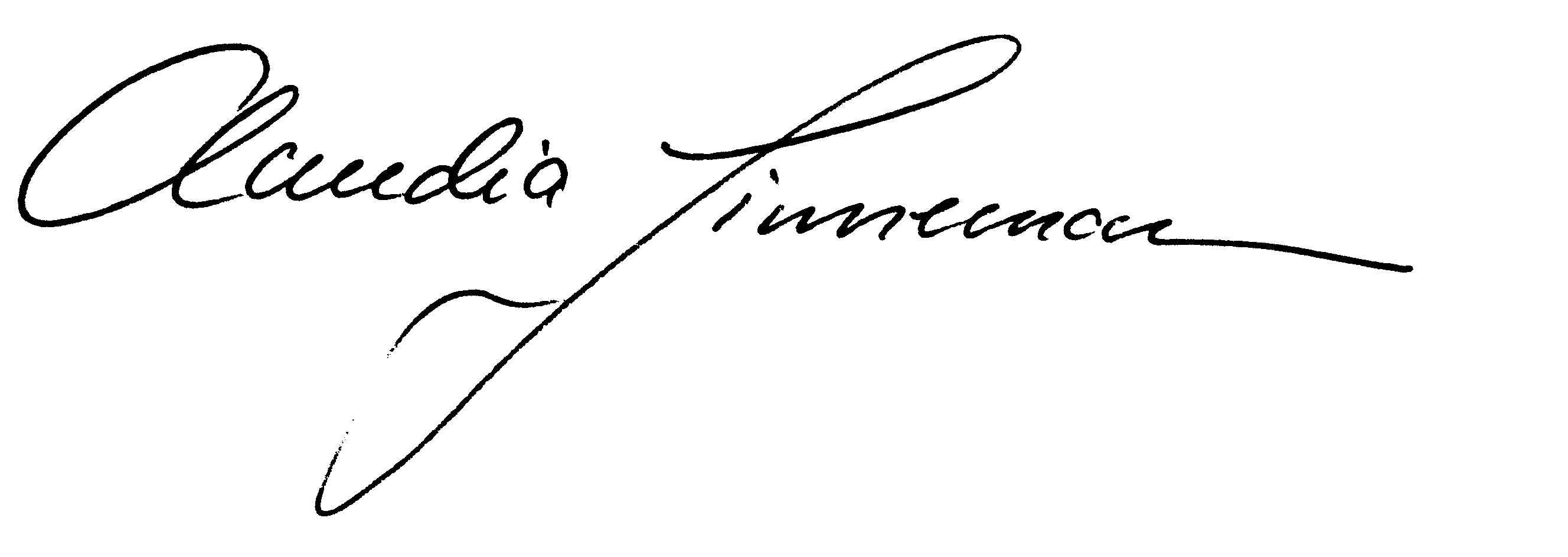


## Appendix 2



## Declaration of Authenticity

I hereby declare that the work submitted is my own and that all passages and ideas that are not mine have been fully and properly acknowledged.



Mels, 6.1.2020