**Programming three variations of chess   
Taking a chess code and newly programming it to make three unique variations of chess,**

****

**Matura Paper, Kantonsschule Sargans**

Attila Vizhanyo 4bNP

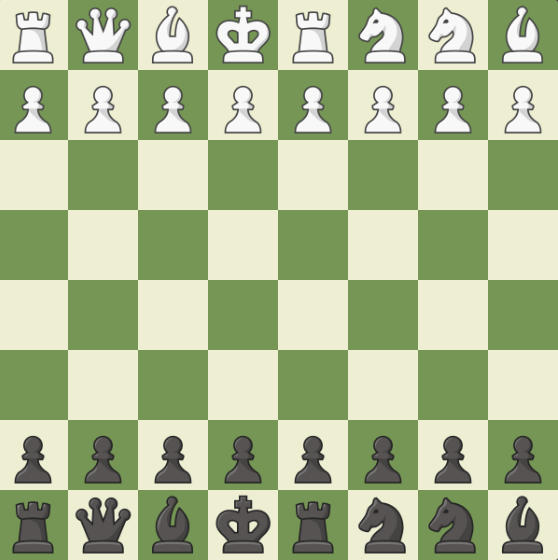
**Supervisor:**Herr Lukas Fehr

**Submitted on:**  
6 January 2023

# 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.

I 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 in order to make chess a more enjoyable game.

# 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 Original Chess Code 2](#_Toc454175654)

[3 Color Chess 2](#_Toc454175654)

[3.1 Rules of Color Chess 2](#_Toc454175655)

[3.2 Coding of Color Chess 2](#_Toc454175656)

[4 Chaotic Chess 3](#_Toc454175657)

[4.1 Rules of Chaotic Chess 3](#_Toc454175658)

[4.2 Coding of Chaotic Chess 3](#_Toc454175659)

[5 Elemental Chess 4](#_Toc454175660)

[5.1 Rules of Elemental Chess 4](#_Toc454175661)

[5.2 Coding of Elemental Chess 4](#_Toc454175662)

[6 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

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.

The next question that came up regarding my project, was how many chess variants I wanted to program. The answer was not straightforward because I first needed to figure out what the rules for my chess variations would be to estimate the effort required to code each variation. In the beginning, I presented two variations, named “Color Chess” and “Chaotic Chess” to my supervisor. After he inspected the rules of the two game-modes, he informed me that he liked the ideas, and I should come up with a third variation to fulfill the requirements of a diploma work. Subsequently, I figured out a third variation that involved around the four elements of the world, “elemental chess”.

When coming up with the different variations of chess, I had to ask myself which aspects of chess do I want to modify, and which ones do I want to keep? I did not want to reinvent the game, therefore, I preserved the fundamentals characteristics of chess, by not changing the board, pieces, and movements of the pieces. I mainly focused on adding special abilities and items to make the game feel more like a video game.

(??) Because of these features, my variations are best suited for an electronic device since displaying them on a standard chess board would be difficult.

---

For the first variant I took inspiration from a modern-day game called “Splatoon”. The objective of the game is to cover as much area as possible with a color that is assigned to you. To achieve victory your team must color more area than the opponent team. This game concept is not unique however, there are many similar games that are based on the same idea. I determined that his game idea can be well adapted onto chess. By assigning a color to each square that a white piece moves onto, and another color for which squares a black player moves onto, the two players can compete for the control over the coloring of the squares on the chessboard.

Items are a core part of video games. Most games have some sort of item that does a special action inside of the game. In video games like Super Mario you have items all over the world that make you stronger. In chess however there aren’t any items you can interact with. So, my goal with my second variant was to implement four different items that could be picked up by chess pieces, to intensify the in-game experience.

With my third variant I wanted to incorporate the four elements of the world into chess, earth, water, air, and fire.

## Procedure and Method

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 my to save my projects in a history, which is crucial in programming. When writing any code it is important to make many back ups and saves in order to jump back to the code if a long term error or mistake happens. GitHub

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 supports a wide range of programming languages with loads of extensions. I found using visual studio practical because of it’s simplicity and low learning curve.

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.

With my knowledge about the underlying structure of the system I could now work on programming the first variant “Color Chess”.

## Structure of the Paper

# Original Chess Code

Process of original chess code

Ein Bild, das Text, Screenshot, Schrift, Design enthält.

Automatisch generierte BeschreibungIn the first step, the program sets the turn to white, since in the game of chess white always plays the first move. The program then proceeds by creating an eight-by-eight field that represent the chess board. Afterwards, the position is set up by adding the chess pieces to their corresponding starting location.

Next the program waits for an input of the user. This will be a click on a field that the user wants to move a piece from. The program must now check if the selected field does indeed have a white piece on it. By taking the value of the field, the program can determine if there is a piece on the selected field. If the field is empty, the user must select a field again. If the user succeeds in picking a field with a piece on it, it colors that field green, to indicate that it has been selected. Afterwards, the program must calculate every possible move with the selected piece and writes down coordinates to which fields the piece can travel to. These fields will be marked with a green circle to show the user, what moves are available to him.

Then the computer waits for the second input of the user that will let the computer know to which field to move the selected piece to. If the destination field corresponds to the possible moves of the selected piece, the program allows the move to be played. If not, the program asks the user to select a piece again and make another move.

Once the move was successful, the turns switch and black is asked to select one of his pieces.

## Important functions and variables explained

The following list concludes crucial functions and variables of the original chess code that purpose is necessary to understand in order to understand the coding of the variations.

**Buttons**: The chess board is made up by an eight-by-eight field of buttons. I liked the usage of buttons to represent the square of the chess board, since each button has a distinct identity, and it creates an event when it is clicked on. These values help determine which square the user has selected and where they intend to move.

**startbutton**: This is the button that the user presses on the first time to select a piece of their own, that they want to move.

**endbutton**: This is the button that the user presses on the second time, to which button they want to move the selected piece on.

**turn**: This is the label that determines which player is next to move. A value gets passed onto the turn either “W” for white or “B” for Black.

**Check\_if\_move\_legit**: This is the functions that determines if the move from the startbutton to the endbutton is a legit move of chess.

**Update\_possible\_moves**: This is the functions that updates every possible move that can be made after a move is played. Since after every move a new position of the chess board is portraid, the program also has to adapt to this new environment and understand which moves are valid to play.

**btnClick**:

**btnID**: This marks the button green the player selected with startbutton.

**undo\_coloring**: This erases every lightgreen marked button that the piece could move onto.

**Figure**: This family gives each chess piece a name, it’s position and it’s moves that it can make from that position

Ps: Stand for position start, this is the square that the piece is represented on. If a piece is captured by another, it gets the board position (-1/-1), which means that piece isn’t on the board anymore.

# Color Chess

## Rules of Color Chess

In "Color Chess," the squares to which the pieces move are highlighted with color, with each player assigned a unique color. The goal of the game is to color in more square with your own color than your opponent. Only the square that the chosen piece steps or has been standing on will be colored in. Since each game of chess starts with the same position, the first two rows on each side will be colored in.

Starting position:

\*\*\*img\*\*\*

If a your piece captures the opponents piece, your piece will overtake their colored square and the program will highlight that square with your own color. Because of this, it is more advantageous to capture your opponents pieces instead of coloring in empty fields.

There is a limited number of rounds that can be played before the game ends. The objective is to color in as much squares with your own color within this limited rounds period as possible. After 30 rounds (a round consist of one move of a piece, so two rounds would be each player making a move) the games ends automatically and the computer evaluates how many squares each player has colored in. The player that colored in more proceeds to win.

The traditional way of winning in chess is cooperated into the game aswell, so you can win instantly by setting your opponent into checkmate.

## Coding of Color Chess

# Chaotic Chess

## Rules of Chaotic Chess

In the game of Chaotic Chess there are four items that modify the common rules of chess. These items are randomly distributed on the fields of the chess board. Pieces can interact with the item by stepping on the field that are occupied by the item.

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

**Bomb:**

Picking up the bomb induces an explosion of a three-by-three area. Every piece in that area is defeated and removed from the bord. This way clever sacrifices can be made by sacrificing your own pieces to take down more and stronger pieces of your opponent. The bomb can blow up every piece on the board, so checkmate can attain, by stepping on a bomb near the king. The king himself can step onto the bomb as well, which would lead to a victory of the enemy team. 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 Example of a bomb explosion at the edge

**Ein Bild, das Brettspiel, Schachfigur, Spiele, Tabletopspiel enthält.

Automatisch generierte BeschreibungEin Bild, das Schachfigur, Brettspiel, Schach enthält.

Automatisch generierte BeschreibungEin Bild, das Schachfigur, Schach, Brettspiel, Tabletopspiel enthält.

Automatisch generierte BeschreibungCoin:**

The piece that stepped on the coin, gets converted into the piece that is resembled on 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.

Ein Bild, das Schachfigur, Schach, Brettspiel enthält.

Automatisch generierte BeschreibungEin Bild, das Schachfigur, Brettspiel, Tabletopspiel, Spiele enthält.

Automatisch generierte Beschreibung Example of a coin usage

**Shield:**

The shield makes the piece that stepped on to the shield invincible for three rounds. The shield is anchored on its field and does not move with the piece that picked the shield up. Every piece can pick up the shield.

Example of a shield usage

Ein Bild, das Schachfigur, Brettspiel, Schach enthält.

Automatisch generierte BeschreibungEin Bild, das Brettspiel, Schach, Schachfigur, Spiele enthält.

Automatisch generierte Beschreibung

**Barrier:**

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

Jump over the barrier. Check mate with the barrier.

Ein Bild, das Brettspiel, Schachfigur, Schach enthält.

Automatisch generierte BeschreibungEin Bild, das Schachfigur, Brettspiel, Schach, Spiele enthält.

Automatisch generierte Beschreibung

## Coding of Chaotic Chess

**Basic properties of an item**

To explain what each item individually does; I will first describe what these items have in common. Because they all have the same underlying code for the generation, visualization, interaction and reappearance of an item.

Generating the position of items randomly

Ein Bild, das Schachfigur, Brettspiel, Tabletopspiel, Spiele enthält.

Automatisch generierte BeschreibungIf I wanted chaotic chess to be a balanced game, I had to generate the items equally on both sides. To make this happen I declared two list of buttons. A button represents a field on the chess board. The white buttons only featured buttons from rows 1-4 and the black buttons rows 5-8. With these two lists, I could now tell the program to randomly select a button from each list, and on each side an item would appear. However, the problem would occur that the program selects a button that is already occupied by a chess piece. This way the item would overtake the piece and delete it. Since I only wanted the items to spawn on empty fields, I had to tell the program to make a random selection of the empty fields on each side. Luckily in the original program two lists called “position\_white\_players” and “position\_black \_players” were available. These lists contained the pieces of white and black and most importantly, their positions. This way I could tell the program that it should randomly choose a button to put the item on, but if that button is already occupied by a chess piece, it should randomly choose another button to place the item on.

White button list

Black button list

  while True:

    bomb\_button = random.choice(button\_list)

    bomb\_pos = getButtonPosition(bomb\_button)

    if bomb\_pos not in players\_pos\_list and bomb\_pos != endbutton\_pos:

      break

Embodiment of an item

Once the item has found a button to be placed on it should store the buttons position, since we will need to fall back on it in the future. Afterwards the item should be visually displaying its location to let the user now where it is. For the sake of simplicity and testing I started off by representing the item with a unique letter and if the item was on the white side, it was upper-case if it was on the black side, it was lower-case. Items on the white side were labelled with “\_1” and items on the black side were labelled “\_2” at the end of the item’s name.

With the command button.config(text = x ) I could edit the text that appears on the button. Since we stored the position of the randomly selected button earlier, we can edit the button that the item is placed on.

BOMB\_1 = "O" # white side bomb

# Bomb\_button is the randomly selected button, where we will place our bomb item on

# We visualize our bomb by editing the buttons text to the bomb symbol

bomb\_button.config(text=BOMB\_1)

Picking up an item

Next, we want to let the program now if the user picked up an item. For this to work, we can use our “endbutton\_text” variable again. By saying that if the “endbutton\_text” is equal to the item symbol, it should know that the user stepped on the item. Inside of this condition, we will code what each item will do if it gets picked up.

  # If text of destination field and text of bomb is the same

  if endbutton\_text == BOMB\_1 or endbutton\_text == BOMB\_2:

    # Let the user now that he picked up the bomb

    action\_label.config(text = "YOU PICKED UP A BOMB!", fg = "black")

Regenerating position of items randomly

With the next step I wanted to make the game more random and chaotic, so I decided to change the position of the items after a certain number of rounds. To make this happened I had to delete the original items place if it already existed and choose another position for it. We should also note that if the item has already been picked up it should not delete its former place since the player already removed the item from the board by picking it up. However, this code does not apply to the barrier, since it’s impossible to pick up the barrier and we always have to delete it’s former position. In order to delete the former item, we can make use of button.config(text = “”). The (“”) indicate to the computer that it should display a string, which is simply a text. By not entering anything between the (“”) it tells the computer it should edit the buttons text to nothing.

**Bomb**

When the bomb is picked up it has to color the fields surrounding the bomb black. In order to achieve this 3-by-3 black area I took use of the nested for loop function.

**Coin**

One unique feature about the coin is that it generates with a random choice of text. To get a random piece represented on the coin, I had to use the random module by importing it with “import random”. Now I could tell the program to make a random choice between three different figures including bishop, knight, and rook.

COIN\_LIST = [BB, BN, BR]

coin\_symbol = random.choice(COIN\_LIST)

With this code our coin item will pick a random symbol in coin list and use that symbol to be represented on the coin. Next up the program has to know if the coin is picked up, which symbol is represented on it in order to transform the piece that stepped on the field into the piece on the coin. For this I used “if functions”. At first the program has to determine which player stepped on the coin. In the original chess code, there was already a function that determine which players move it is. So if the coin is picked up when white is on the turn, the code should use the white pieces to transform the piece into the coin. Then when the program nows which color stepped on the coin, it should read what symbol is on the coin. We can use the endbutton\_text again. So if the endbutton\_text is the rook and the turn is white, it should turn the piece that stepped on the coin into a white rook.

  elif player\_color == 'W':

    if endbutton\_text == BB:

      endbutton.config(text = WB)

      figure.name = 'WL'

    if endbutton\_text == BN:

      endbutton.config(text = WN)

      figure.name = 'WS'

    if endbutton\_text == BR:

      endbutton.config(text = WR)

      figure.name = 'WT'

However, there is the same problem that occurs like with the bomb. The program only replaces the look of the piece but not it’s properties. So, although our piece now transformed into the coin piece, it still moves and behaves like before stepping on the coin. Therefore we have to now tell our code that the piece that stepped on the coin should not only take the symbol of the coin piece, but also take the values and properties of coin piece.

**Shield**

**Barrier**

The last item is the barrier. With the help of the code that was already in the original chess program, the coding of this item was simpler. To make the barrier work I had to use the function “check\_if\_move\_legit”, which calculates if a move is possible and the variable “endutton\_text”, which stores the text of the button that the user clicked on the second time. Within the “check\_if\_move\_legit” function, I coded that if the “endbutton\_text” is equal to the barrier’s text, the program should tell the user he can’t step onto a barrier and not allow this move by returning “False”. This then leads to an error and the program ask the user to play another move.

  def check\_if\_move\_legit(self, ps, pe, all\_positions\_enemy, all\_positions\_own\_team, all\_possible\_moves\_enemy, turn):

    # if user chooses barrier as it's destination field,

    if endbutton\_text == barrier\_text:

      # tell the user it cannot step onto the barrier

      action\_label.config(text = "YOU CAN'T STEP ONTO A BARRIER", fg = "red")

      # makes the move impossible to make

      return False

# Chapter 4

## Subchapter 4

## Subchapter 4

# Summary

# Bibliography

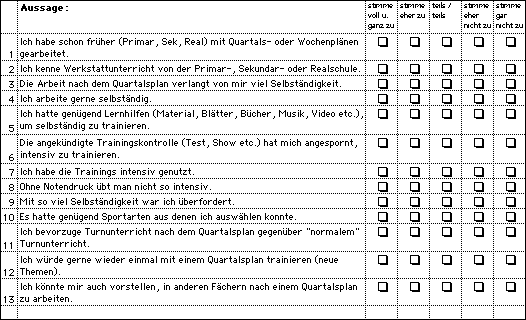
Fisher interview: https://qualiacomputing.com/2022/07/25/bobby-fischer-1972-world-chess-champion-on-why-chess-is-a-lousy-game-and-how-to-save-it/

# Table of Figures

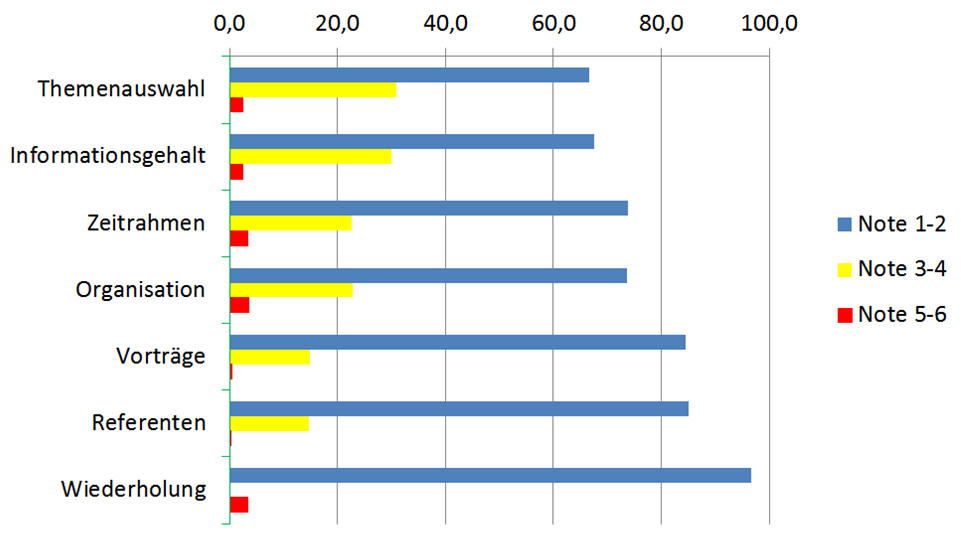
[Figure 1: Demographics of Switzerland in 1900 and 2014 3](file:///D:/Sargans%202016-2017/3-Info/9)%20Maturavorlage%20erstellen/Kurs%20MA-Vorlage%20Reuteler/Vorlage%20Maturaarbeit%20Englisch.docx#_Toc454176099)

# Appendix

## Appendix 1

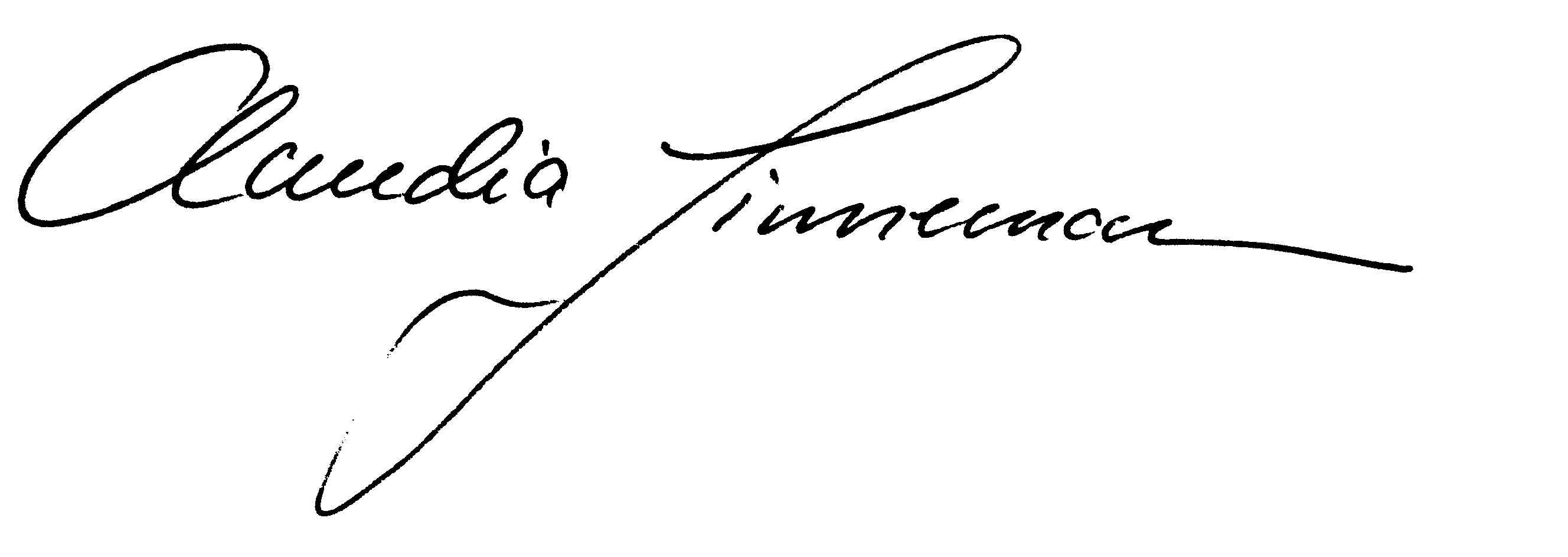


## Appendix 2



## 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