**The Coding of Chess Variations  
Implementing Different Ways to Play Chess Into a Standard Chess Program**



**Matura Paper, Kantonsschule Sargans**

Attila Vizhanyo, 4bNP

**Supervisor:**Herr Lukas Fehr

**Submitted on:**  
8 January 2024

# 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, whose respond was the following to an interview question:

Figure 1 Chess 960 Starting Position

“**Interviewer**: Why do you hate Chess? Being the be… probably, possibly, the best Chess player ever?

**Bobby**: Because I know what Chess all is about! It’s all about memorization. It’s all about pre-arrangement…” (algekalipso, 2022)

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 Contents

[Preface III](#_Toc155112882)

[Table of Contents IV](#_Toc155112883)

[1 Introduction 1](#_Toc155112884)

[1.1 Objectives and Guiding Questions 1](#_Toc155112885)

[1.2 Procedure and Method 1](#_Toc155112886)

[Structure of the Paper 2](#_Toc155112887)

[1.3 2](#_Toc155112888)

[2 Initial Chess Code 2](#_Toc155112889)

[2.1 Logic of initial chess code 2](#_Toc155112890)

[2.1.1 Overview 2](#_Toc155112891)

[2.1.2 Control Flow 4](#_Toc155112892)

[2.1.3 Error Handling 4](#_Toc155112893)

[2.2 GUI 5](#_Toc155112894)

[2.2.1 Overview 5](#_Toc155112895)

[2.2.2 Visualization Chess Pieces 5](#_Toc155112896)

[2.2.3 Visualization Chessboard 5](#_Toc155112897)

[2.2.4 Update Game after a move 6](#_Toc155112899)

[2.3 Function catalog 6](#_Toc155112900)

[main: Regulates the entire process of the chess program, besides setting up variables, buttons and the pieces. 6](#_Toc155112901)

[update\_all\_possible\_moves: Use the current board to see which moves can be played according to the rules of chess. 6](#_Toc155112902)

[checkinput: Checks whether the move played puts the player who made the move in check. 6](#_Toc155112903)

[check\_chosen\_move: 6](#_Toc155112904)

[print\_board: Prints the current state of the board to the terminal. 6](#_Toc155112905)

[get\_position: Stores position of each piece on the board. 6](#_Toc155112906)

[3 Color Chess 7](#_Toc155112907)

[3.1 Rules of Color Chess 7](#_Toc155112908)

[3.2 Implementation of Color Chess 8](#_Toc155112909)

[4 Chaotic Chess 9](#_Toc155112910)

[4.1 Rules of Chaotic Chess 9](#_Toc155112911)

[4.1.1 Barrier 9](#_Toc155112912)

[4.1.2 Shiel 10](#_Toc155112913)

[4.1.3 Coin 11](#_Toc155112914)

[4.1.4 Bomb 12](#_Toc155112915)

[4.2 Implementation of Chaotic Chess 13](#_Toc155112916)

[4.2.1 Overview 13](#_Toc155112917)

[4.2.2 Common code for items 14](#_Toc155112918)

[4.2.3 Barrier 16](#_Toc155112919)

[4.2.4 Shield 16](#_Toc155112920)

[4.2.5 Coin 16](#_Toc155112921)

[4.2.6 Bomb 16](#_Toc155112922)

[5 Summary 17](#_Toc155112923)

[Bibliography IV](#_Toc155112924)

[Table of Figures V](#_Toc155112925)

[Appendix VI](#_Toc155112926)

[A. Appendix 1 VI](#_Toc155112927)

[B. Appendix 2 VII](#_Toc155112928)

[C. Declaration of Authenticity VIII](#_Toc155112929)

# Introduction

## Objectives and Guiding Questions

With my chess variations I aim to create a game that is visually represented on a display. I want to create a program that check the legitimacy of each chess move and prevent it, if the move is not valid. I aim to enforce the rules of the game, which include my two chess variations.

Compute the legit moves of each piece. Visualize the game.

Enforce the rules of the game, which includes…

## 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

In this paper, I will present two chess games which are variations of the standard chess.

Through my project, I aim to guide you in creating two chess variants. Whether or not you have experience with coding, my goal is to present this information in a way that is understandable for everyone.

In the first chapter I will explain the underlying chess program which I took “as a starting point” for my project. Understanding the original chess program is required to be able to understand the rest of my work. I will break down the program into its logic and display. I will describe its functions and variables, and explain how the game is visualized.

In the second chapter…

# Initial Chess Code

## Logic of initial chess code

### Overview

Concept

#### Figure (or Piece)

A Class is like an object constructor, or a "blueprint" for creating objects. (W3Schools, s.d.)

The first key concept that needs to be modeled in any chess program is the concept of the chess pieces, also called the figures. There are sixteen pieces of each color, black and white: one king, one queen, two rooks, two bishops, two knights, and eight pawns. Each piece has a type and a position. Each piece has a defined starting position at the beginning of the game. During the game, the players move the white and black pieces in alternate turns, thus the position of the pieces need to be updated. The pieces can move to a square that is either unoccupied or occupied by the opponent’s piece. Each figure needs to know which squares it can move to, based on its type.

A white paper with black text

Description automatically generated

|  |  |  |
| --- | --- | --- |
| Member name | Description | Comments |
| name |  |  |
| object\_name |  |  |
| color | Unicode characters to display the image of the chess piece. |  |
| position | The current coordinates of the piece. | e.g. (0,4) – see Section 2.1.1.3 |
| possible\_moves | The coordinates to which the piece can move. |  |

#### Chessboard

Chess is played on a square board of eight rows and eight columns.

#### Position

The squares of the board and the chess pieces have a separate grid to arrange them. The grid for the buttons ranges from row and column one to eight. Whereas the grid arranging the pieces range from row and column zero to seven. This is because computers start counting from zero. So, when comparing the two grids, we must subtract one from the grid of the buttons to correspond to the grid utilized by the pieces.

A screenshot of a game

Description automatically generated

Figure 2 This is the grid used by the buttons.

#### A game of chess with a checkerboard and chess pieces Description automatically generated

Figure 3 This is the grid used by the pieces.

In the code the chess pieces are stored as individual objects of the Figure class. If we want to interact with the piece, we must call them by their name. The pieces are named the following way.

#### A green and white checkered board with numbers and letters Description automatically generated

Figure 4 The pieces are named this way.

#### Chessboard

#### Turn

In the game of chess, each player makes the move one after the other.

Turn is a variable that the program uses to determine which player is making the next move. This variable is important to check whether the move played is legitimate. The turn variable can be set to “W” (for white) or “B” (for black). If a legit move has been played, the Turn must change to the other value. If the player does not make a valid move, the program must not change the value of this variable.

#### Capture

When a figure has been captured in chess it is removed from the broad. Similarly in this program, captured pieces are given the position (-1,-1). The user can no longer interact with pieces that are in this location.

#### Check

A check is given when a piece directly attacks the king. If the program finds a legit move that can capture the king with the next move, it detects a check. When a player is in check, they are asked to play a move that escapes the check.

#### Checkmate

Once a king is in check and the program cannot find a legitimate move to escape the check, it is checkmate and the game is over.

### Control Flow

### Error Handling

A well-written code must deal with its errors. If an error occurs, the program freezes and the user can no longer interact with it. This leaves the user with no idea what caused the problem. To prevent this, the original chess code introduces a variable called “error”. This variable is used to tell the user what problem occurred while interacting with the program. The program continues to run and since the user is informed about the problem, he can avoid it. The error variable can be equal to four values.

If the error is set to "1", the provided move was invalid.

When the error equals “2”, the game is over.

In the case where the error is “3”, the king is in check.

Should the error be equal to “4”, there is a checkmate.

## GUI

### Overview

Tkinter

### Visualization Chess Pieces

The chess pieces displayed on the GUI are represented by Unicode Characters.

Unicode is a universal character set that defines all the characters needed for writing the majority of living languages in use on computers. (W3Schools, s.d.)

The Unicode characters are stored within string variables. Every identical chess piece of the same color has the same Unicode. There are a total of twelve Unicode characters needed to represent the black and white pieces: pawn, knight, bishop, rook, queen and king, for both colors.

We can put a chess piece on a button by configuring the button to contain the text of a selected Unicode. This way we can set the starting position of the chess game by setting the text of the buttons.

A black background with a black square

Description automatically generated with medium confidence

Figure 5 '|u265C' Unicode character for a black rook

BR = '\u265C'

a8 = Button(tk, text=BR, font='Times 20 bold', bg='white', height=2, width=5, command=lambda: [btnClick(a8), btnID('a8')])

### Visualization Chessboard

The chessboard is made up of an eight-by-eight field of squares. A square on the board is represented by a button in the program. “A button is a tk widget which is designed for the user to interact with, i.e. if the button is pressed by mouse click, some action might be started. They can also contain text and images like labels.” (Klein, 2022). A grid is used to arrange the buttons horizontally and vertically into a table format, where each button has its own coordinates. Each button is stored in a variable. \*\*\*button class To create a button, "Button()" is used, with its properties contained inside the brackets. To display our button on the board, we must attach it to the grid by specifying its row and column.

a8 = Button(tk, text=BR, font='Times 20 bold', bg='white', height=2, width=5, command=lambda: [btnClick(a8), btnID('a8')])

a8.grid(row=1, column=1)

### Update Game after a move

To make an ongoing chess game, the program must update the position of each piece. It is also necessary to calculate every possible move that can be played from the current position. And finally, the program needs to display the updated board.

When the user clicks on the piece he wants to move with, the program saves the text of its button. Once the user selects a legitimate button to move the piece to, the program can transfer the saved text to display on the target button.

## Function catalog

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. (W3Schools, s.d.)

btnClick:

undo\_coloring: Inverts the highlighting of squares that can be moved to when a figure is selected.

btnID:

## main: Regulates the entire process of the chess program, besides setting up variables, buttons and the pieces.

## update\_all\_possible\_moves: Use the current board to see which moves can be played according to the rules of chess.

## checkinput: Checks whether the move played puts the player who made the move in check.

## check\_chosen\_move:

## print\_board: Prints the current state of the board to the terminal.

## get\_position: Stores position of each piece on the board.

# 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.

A screenshot of a game

Description automatically generated

Figure 6 Color Chess Starting Position

A screenshot of a game

Description automatically generated

Figure 7 Color Chess Piece Capture Example Prior

A screenshot of a game

Description automatically generated

Figure 8 Color Chess Piece Capture Example After

## Implementation of Color Chess

# 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.

A screenshot of a game

Description automatically generated

Figure 9 Chaotic Chess Random Position Example

### 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.

A screenshot of a game

Description automatically generated

Figure 10 Chaotic Chess Barrier Usage Example Prior

A screenshot of a game

Description automatically generated

Figure 11 Chaotic Chess Barrier Usage Example After

### Shiel

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.

A screenshot of a game

Description automatically generated

Figure 12 Chaotic Chess Shield Usage Example Prior

A screenshot of a game

Description automatically generated

Figure 13 Chaotic Chess Shield Usage Example After

### Coin

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.

A screenshot of a game

Description automatically generated

Figure 14 Chaotic Chess Coin Usage Example Prior

A screenshot of a game

Description automatically generated

Figure 15 Chaotic Chess Coin Usage Example After

### 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.

A screenshot of a game

Description automatically generated

Figure 16 Chaotic Chess Bomb Usage Example Prior

A screenshot of a game

Description automatically generated

Figure 17 Chaotic Chess Bomb Usage Example After

## Implementation of Chaotic Chess

### Overview

#### Modules

“Consider a module to be the same as a code library. A file containing a set of functions you want to include in your application.” (W3Schools, s.d.)

To make my program easier to maintain and easier to read, I have put the code of each item into a separate module.

Importing the four modules (barrier.py, bomb.py, coin.py, shield.py) into the initial chess code allows you to use the functions contained in the modules. However, it is not allowed to import the main module into from the modules to your main code for efficiency.

import bomb

import coin

import shield

import barrier

#### Differences In Chess Code

#### Usage Of Initial Chess Code

The item differs in the code that they execute when they are picked up. Otherwise they share similar code.

### Common code for items

#### Placing an item

To find a location where the item can be placed on, we need to randomly select a square on the board.

In the result of developing Chaotic Chess in a balanced way, I had to generate the items equally on both sides of the board. To accomplish this, I split the board in half and declared two list of buttons, representing the squares of the board on each side. Buttons on the white side (row: one to four) and buttons on the black side (row: five to eight).

A chess board with chess pieces

Description automatically generated

With these two lists, I can write a program that randomly selects a button from each list. Leaving us with two buttons that our item can be placed on. However, there is a problem if the code selects a button which is already occupied by a chess piece. This would lead to overwriting the button and deleting the piece. To prevent this, I had to tell the program to select a different button when it is occupied. To determine whether a button is occupied, I used a list that was already included in the original chess code. “Player\_pos\_list” contains the positions of all chess pieces on the board. Using this list, I created a loop in which the program randomly selects a button until it finds an unoccupied one. Once it finds a suitable button, it should then store its variable, since we will need to come back to it.

  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

\*\*\*save item button

#### Visualization of an Item

To let the user know where the item has been placed, it needs to be displayed on the screen.

For the visualization of the items I chose to use Unicode characters. Displaying the item onto a square, works the same way as with the pieces, i.e. by configuring the text of the relevant button. Another advantage of using Unicode is that it can be colorized. I colored the items the following way:

(barrier: purple), (shield: red), (bomb: green), (coin:yellow)

A black background with a black square

Description automatically generated with medium confidence

Figure 19 Barrier Unicode \U0001F5D9

A black background with a black square

Description automatically generated with medium confidence

Figure 20 Shield Unicode \U0001F6E1

A black background with a black square

Description automatically generated with medium confidence

Figure 21 Bomb Unicode \U0001F4A3

For the visualization of the Coin, I used the Unicode character of the corresponding chess piece.

#### Picking up an item

Picking up an item means, that the player moves a chess piece to the field where the item is located.

When the item is picked up it must execute a function, depending on what the item is.

The end button is the button to which the user wants to move the selected chess piece. To tell the program which item has been picked up, the endbutton and the button of the specific item need to be the same.

If the user makes a legitimate move onto an item, we want to execute the function of the item.

Next, we want to let the program know 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")

#### Change position of an item

By changing the position of an item, I want to make the game more random and chaotic. 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.

### Barrier

### Shield

### Coin

### Bomb

# Summary

# Bibliography

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

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

*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

algekalipso. (25. July 2022). *Qualia Computing*. Von Qualia Computing: https://qualiacomputing.com/2022/07/25/bobby-fischer-1972-world-chess-champion-on-why-chess-is-a-lousy-game-and-how-to-save-it/#:~:text=Bobby%20Fischer%3A%20No%2C%20no%2C,I%20don't%20need%20it. abgerufen

*W3Schools*. (kein Datum). Von https://www.w3schools.com/python/python\_classes.asp abgerufen

*W3Schools*. (kein Datum). Von https://www.w3schools.com/python/python\_functions.asp abgerufen

*W3Schools*. (kein Datum). Von https://www.w3schools.com/charsets/ref\_html\_utf8.asp abgerufen

Klein, B. (1. 2 2022). *python-course*. Von https://python-course.eu/tkinter/buttons-in-tkinter.php#:~:text=The%20Button%20widget%20is%20a,text%20and%20images%20like%20labels. abgerufen

*W3Schools*. (kein Datum). Von https://www.w3schools.com/python/python\_modules.asp abgerufen

# Table of Figures

[Figure 1 Chess 960 Starting Position III](/Users/attila/Documents/GitHub/matura-arbeit/matura-paper-template.docx" \l "_Toc155112930)

[Figure 2 This is the grid used by the buttons. 3](#_Toc155112931)

[Figure 3 This is the grid used by the pieces. 3](#_Toc155112932)

[Figure 4 The pieces are named this way. 4](#_Toc155112933)

[Figure 5 '|u265C' Unicode character for a black rook 5](#_Toc155112934)

[Figure 6 Color Chess Starting Position 7](#_Toc155112935)

[Figure 7 Color Chess Piece Capture Example Prior 7](#_Toc155112936)

[Figure 8 Color Chess Piece Capture Example After 8](#_Toc155112937)

[Figure 9 Chaotic Chess Random Position Example 9](#_Toc155112938)

[Figure 10 Chaotic Chess Barrier Usage Example Prior 9](#_Toc155112939)

[Figure 11 Chaotic Chess Barrier Usage Example After 10](#_Toc155112940)

[Figure 12 Chaotic Chess Shield Usage Example Prior 10](#_Toc155112941)

[Figure 13 Chaotic Chess Shield Usage Example After 11](#_Toc155112942)

[Figure 14 Chaotic Chess Coin Usage Example Prior 11](#_Toc155112943)

[Figure 15 Chaotic Chess Coin Usage Example After 12](#_Toc155112944)

[Figure 16 Chaotic Chess Bomb Usage Example Prior 12](#_Toc155112945)

[Figure 17 Chaotic Chess Bomb Usage Example After 13](#_Toc155112946)

[Figure 18 Chaotic Chess Modules 13](#_Toc155112947)

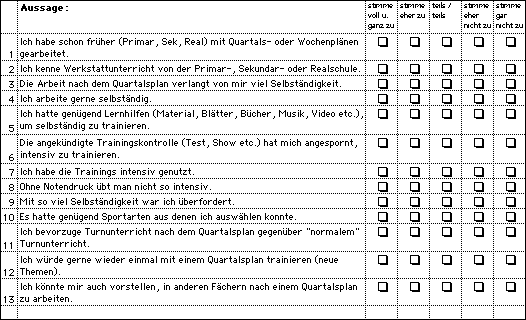
[Figure 19 Barrier Unicode \U0001F5D9 15](#_Toc155112948)

[Figure 20 Shield Unicode \U0001F6E1 15](#_Toc155112949)

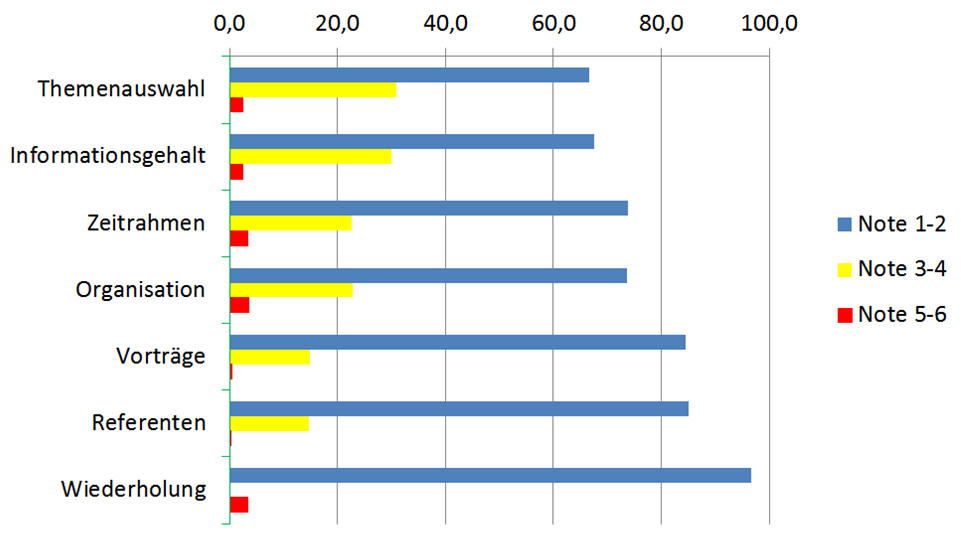
[Figure 21 Bomb Unicode \U0001F4A3 15](#_Toc155112950)

# Appendix

## Appendix 1

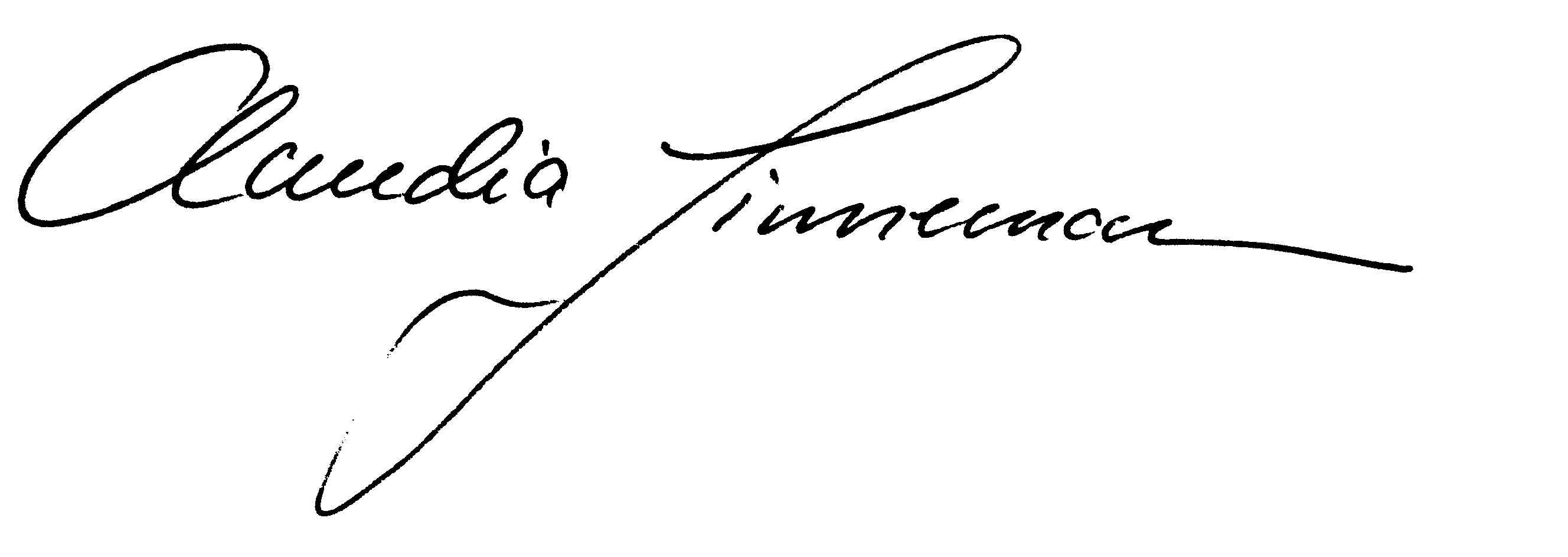


## Appendix 2



## 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