DELTA – Střední škola informatiky a ekonomie, Základní škola a Mateřská škola, s.r.o.
Ke Kamenci 151, PARDUBICE

MATURITNÍ PROJEKT
ANIMACE PRINCIPŮ HISTORICKÝCH KRYPTOGRAFICKÝCH ALGORITMŮ

Ondřej Skutil
4.B
Informační technologie 18-20-M/01
2022/23

Zadání maturitního projektu z informatických předmětů

Jméno a příjmení: Ondřej Skutil

Pro školní rok: *2022/2023*

Třída: *4.*

Obor: *Informační technologie 18-20-M/01*

Téma práce: Animace principů historických kryptografických algoritmů

Vedoucí práce: Mgr. Josef Horálek, Ph.D.

Způsob zpracování, cíle práce, pokyny k obsahu a rozsahu práce:

Cílem maturitního projektu je vytvořit sadu animací vysvětlujících historické kryptografické algoritmy. V teoretické části práce autor představí možnosti realizace animací v různých grafických nástrojích a na základě definovaných parametrů vybere jejich z popsaných nástrojů pro realizaci. V praktické části autor popíše vybrané historické kryptografické algoritmy a vytvoří ukázkové animace, sloužící pro objasnění dané problematiky. V rámci práce budou zpracovány tyto kryptografické algoritmy:

• Caesarova šifra

• Substituční šifry

• Playfairova šifra

• Homofonní šifrování

Stručný časový harmonogram (s daty a konkretizovanými úkoly):

09 – 10 Analýza možnosti realizace animací v různých grafických nástrojích a na základě definovaných parametrů

11 – 12 Podrobný popis principů vybraných historických kryptografických algoritmů

12 – 01 Návrh a ověření animací

02 - 03 Dokončení praktických řešení

03 Finalizace textového znění maturitního projektu

RESUMÉ

Tato maturitní práce popisuje principy kryptografických šifer. V teoretické části se budu zabývat výběru správného softwaru k provedení praktické části a podrobněji popíšu vybrané šifry. V praktické části ukážu, principy kryptografických šifer pomocí animací.

KLÍČOVÁ SLOVA

GUI; cross-platform; framework; API; multithreading; component; endpoint; Canvas

ANNOTATION

This senior thesis describes the principles of cryptographic ciphers. In the theoretical part, I will discuss the selection of the right software to perform the practical part and describe the selected ciphers in detail. In the practical part I will show the principles of cryptographic ciphers using animations.

KEYWORDS

GUI; cross-platform; framework; API; multithreading; component; endpoint; Canvas

Poděkování

Chtěl bych poděkovat Mgr. Josefu Horálkovi Ph.D. za odborné vedení maturitního projektu a poskytnutí cenné zpětné vazby v průběhu vývoje.

Prohlášení

Prohlašuji, že jsme svou práci vypracoval samostatně a použil jsem výhradně prameny uvedené v citacích.

V Pardubicích 31.3.2023

.....

Ondřej Skutil

Obsah

1 Úvod

1.1 Motivace

V nynější době i v minulosti byly a stále jsou informace to nejcennější. Z čehož vyplývá, že je budeme chtít chránit. Zajímalo mě, jak bezpečně přenášet informace, tak aby třetí strana měla problém s odhalením jejich obsahu. V minulosti nebyla globalizace výrazná. To vedlo k šifrám, které jsou v různých způsobech originální. Proto mně připadají zajímavější. Přece jen šifra, kterou nikdo nezná je nejbezpečnější.

1.2 Cíl

Cílem maturitní práce je podrobně popsat a realizovat vybrané problémy v oblasti kryptografie zaměřených na historické šifry a animace k nim. Tento maturitní projekt bude využit jako doprovodný studijní materiál předmětu algoritmy, jehož cílem je seznámit studenty se základními principy šifrování. Pro naplnění cílů této práce byly principy šifer podrobně popsány. Praktická realizace celého projektu byla navržena v jazyk. Výstupem práce je webová aplikace, která umožňuje uživateli výběr konkrétní šifry a jeho implementaci.

2 TEORETICKÁ ČÁST

2.1 TECHNOLOGIE

2.1.1 VULKAN

je nízko úrovňové grafické API zaměřené na rychlost a kontrolu nad hardwarem, vyvíjené za jako nástupce OpenGL. [2] Vytvořené skupinou Khronos Group. [1] Podporuje multithreading a VR. Podporuje cross-platformní vývoj Windows, Linux i Android. [3]

2.1.2 PYTHON

je vysokoúrovňový interpretovaný programovací jazyk, který se vyznačuje svou jednoduchostí, tedy I rychlostí vývoje v něm. [4][5] Vytvořen v roce 1991. Populárním se stal díky své schopnosti efektivně řešit problémy v mnoha oblastech. Značnou výhodou je velké množství knihoven a frameworků, vytvořené jeho velkou komunitou, které usnadňují řešení různých úkolů. [6]

2.1.2.1 TKINTER

je standardní cross-platformní grafické uživatelské rozhraní (GUI) pro jazyk Python. [7][8]

2.1.2.2 TURTLE

je knihovna, která poskytuje jednoduché prostředí pro kreslení grafiky pomocí virtuálního kreslícího pera. [9][10]

2.1.3 RUST

je moderní programovací jazyk. [11] Představen v roce 2010 Mozillou. [12] Rust byl navržen pro psaní bezpečného, spolehlivého a výkonného kódu, zejména pro operační systémy, webové prohlížeče, herní enginy, databáze a další. [11] Kombinuje vlastnosti jazyků vysokoúrovňových s nízko úrovňovými. Přináší svůj vlastní způsob alokování paměti na základě propůjčování. [12]

2.1.3.1 ICED

je aktivně vyvíjená cross-platformní moderní grafická knihovna pro Rust. Navržena, aby byla jednoduchá a intuitivní pro použití. [13][16] Postavena na rust-gpu, umožňuje použití moderních grafických API, jako je například Vulkan, DirectX 12 a Metal. [14]

2.1.3.2 EGUI

je cross-platformní grafické uživatelské rozhraní pro Rust. Snadné a flexibilní s vysvětlenými příklady použití. [15][16]

2.1.4 FLUTTER

je open source framework pro cross-platformní vývoj od společnosti Google. [17] Má vlastní sestavovací systém, ten umožňuje sestavit aplikaci bez potřeby znovu kompilovat celý kód.[18]

2.1.5 SOLIDJS

je moderní, deklarativní a výkonný JavaScript framework, vytvořený Ryanem Carniatem jako nástupce Reactu, pro tvorbu webových aplikací a uživatelských rozhraní. Framework se zaměřuje na rychlost a efektivitu při renderování komponent v reálném čase a umožňuje vývojářům psát kód, který je snadno čitelný a udržovatelný. [19] Vzhledově se podobá Reactu. [21]

2.1.5.1 Důvod zvolení

Table 1: Porovnání [19][20]

Feature	React	SolidJS
Typescript support	Yes	Yes
Declarative nature	Yes	Yes
Unidirectional data flow	Yes	Yes
JSX first class support	Yes	Yes
Direct manipulation of the DOM	No	Yes
Avoids component re-rendering	No	Yes
Highly performant	No	Yes
Community and ecosystem	Yes	No
Great documentation	Yes	Yes
Scaffolding tools	Yes	Yes
Conditional rendering	Yes	Yes
Server side rendering	Yes	Yes
Concurrent rendering	Yes	Yes

Table 2: Performances compared to vanilla JS, převzato z https://webtips.dev/solidjs-vs-react

Vanilla	1
Solid	1.05
Inferno	1.20
Svelte	1.27
Preact	1.42
Vue	1.54
React	1.93
Angular	1.93

- Solidjs nepotřebuje memoization, aby se vyhnul re renderování komponentů.
- React porovnává změny ve virtuálním Domu oproti reálnému Domu, poté změny zapíše SolidJS toto obchází, zapisuje přímo do reálného.

2.1.6 ASP.NET

je open-source framework od Microsoftu používaný pro skriptování na straně serveru. [22] Je nástupcem ASP. [23] Programátoři mohou realizovat své projekty v jakémkoliv jazyce podporujícím CLR jej volí pro weby a webové aplikace nové generace využívající HTML, CSS a JavaScript. Pomocí této technologie můžete také vytvářet webová rozhraní API. [23]

2.1.6.1 Důvod zvolení

- Oproti Nodejs je multivláknové.
- Je více integrovaný do komerčních cloudů.
- Podporuje více programovacích jazyku.
- C#, jazyk napsaný v Jave a C++, jsem vybral, protože se vyučuje na škole.

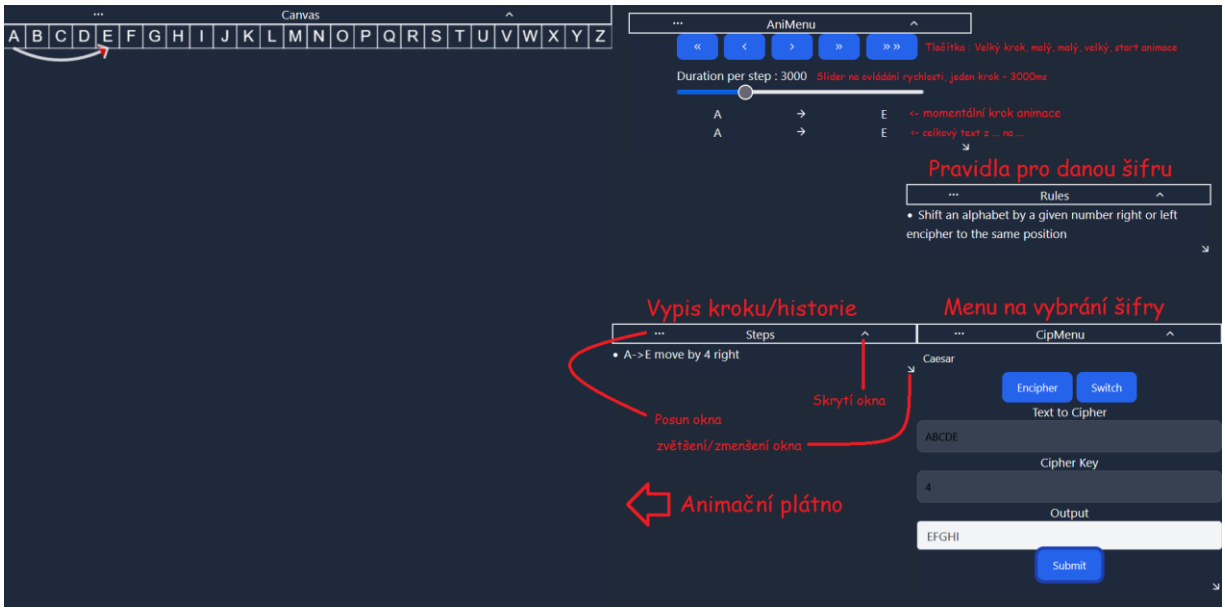
3 PRAKTICKÁ ČÁST

3.1 Popis a realizace aplikace

Aplikace se skládá z pěti různých plovoucích oken, z kterých každé má nějaký význam, dva jsou interaktivní, tím je animační a šifrovací menu, ostatní tři pasivní(animační plátno, pravidla, historie kroků). Šifrovací menu má čtyři povinné vstupy, zpráva, klíč, šifra, mód. Po vyplnění těchto polí uživatel může odeslat informace na server, kde se na daném endpointu zpracují. Vytvoří se, jak výstupový text, tak pole animací. Data putují na frontend, kde se jednotlivé kroky převádějí na pixely. Animace jsou realizované pomocí SET INTERVAL funkce, reference, Promise, javascript closures a Canvas API, responzivita pomocí poměru.

Animační menu nabízí plnou kontrolu nad animací.

HTML5 Canvas jsem si vybral z důvodu „jednoduchosti“. Nabízí jen základní funkce. Zbytek nechává na vývojáři. Při vývoji projektu jsem se potýkal s překážkami jako vykreslování animace, protože re renderování celého canvasu několikrát nepřipadalo v úvahu. Rozhodl jsem se tedy pro „navrstvení“ canvasu. První vrstva pro vzhled šifry a druhý určený čistě pro vykreslování animací. Tímto se zadní musí překreslit jen při zvětšení/zmenšení okna. Další překážkou bylo, zviditelnění principu šifry, to vedlo ke změnění barev a přidání okna pravidla, historie kroků a zobrazování aktuálního kroku šifry v animačním menu.



Picture 1: Vzhled aplikace

3.2 Šifry

3.2.1 Caesarova šifra

Princip Caesarovy šifry je založen na tom, že všechna písmena zprávy jsou během šifrování zaměněna za písmeno, které se nachází o pevně určený počet míst dále v abecedě. [24]
Text: Ahoj
Klíč: 4
A⇒E, h⇒l, o⇒s, j⇒n

Šifrovaný: Elsn

Při dešifrování je postup opačný. Posouvá se doleva.

3.2.1.1 Historie

Tuto šifru používal pro vojenskou komunikaci Julius Caesar a popsal ji v Zápiscích o válce galské (odtud je odvozen její název). Caesar používal posun o tři místa, obecně je ale za Caesarovu šifru označováno jakékoli šifrování na principu prostého posunu písmen (znaků) o konstantní hodnotu. [24]

3.2.1.2 Bezpečnost

Caesarovu šifru lze poměrně snadno vyluštit (tj. pomocí metod kryptoanalýzy zjistit neznámý klíč, což je zde počet míst posuvu v abecedě). Vzhledem k prostému posuvu je v zašifrovaném textu možné odhadnout, některá písmena pomocí odhadu jejich statistického výskytu (což může dále zjednodušit ponechání mezer v šifrovaném textu), tedy použít na zašifrovaný text tzv. frekvenční analýzu. Vzhledem k omezenému (nízkému) počtu možných klíčů je šifra snadno napadnutelná též útokem hrubou silou. [25]

3.2.1.3 Příkladový kód

```
public class CaesarCipher{
static char Caesar(char ch, int key, char d)
{
        // posun o daný počet míst pomocí ASCII, vrácení odpovídajícího charakteru
        return (char)((((ch + key) - d) % 26) + d);
}
public static string Encipher(string input, int key)
{
// Vytvoření proměnné k ukládání upraveného textu
string output = string.Empty;
for(int i = 0; i < input.Length; i++){
        char ch = input[i];
        // Zjištění jestli se vůbec jedná o písmeno
        if (!char.IsLetter(ch)) continue;
        // Zjištění jestli se má posun počítat od ascii hodnoty 65 nebo 97 pro jednotlivé písmeno
        char d = char.IsUpper(ch) ? 'A' : 'a';
// Přidání upraveného písmena k textu
        output += Caesar(input[i], key, d);
}
return output;
}
public static string Decipher(string input, int key)
{
// Zavolání metody zašifrování, ale s posunem doleva
return Encipher(input, 26 - key);
}
}
```

Code 1: Caesarova šifra

3.2.2 Playfairova šifra

Klíčem je mřížka 5×5, 25 písmen abecedně seřazených s odejmutím jednoho písmena (obvykle J). [27] Pokud text obsahuje J, pak je nahrazeno I. [26] Při použití textu jako klíč jsou z textu odebrány speciální znaky, je převeden na velké písmena a poté je upraven, aby se jednotlivý znak vyskytoval jen jednou, následně jsou znaky, které jsou v mřížce posunuty na začátek v pořadí textu, který je použit jako klíč. [26]

Pravidla :

- Pokud je ve dvojici jen jedno písmeno nebo jsou dvě stejná na druhou pozici dát libovolné písmeno (obvykle X), pokračovat dle pravidel.
- Pokud jsou obě písmena ve stejném sloupci, vezměte písmeno pod každým z nich (vraťte se nahoru, pokud je dole.
- Pokud jsou obě písmena ve stejné řadě, vezměte písmeno napravo od každého z nich (vraťte se doleva, pokud je úplně vpravo).
- Pokud neplatí ani jedno z předchozích dvou pravidel, vytvořte obdélník se dvěma písmeny a vezměte písmena na vodorovném protilehlém rohu obdélníku.

Dešifrování šifry Playfair je stejně jednoduché jako provedení stejného procesu obráceně. Přijímač má stejný klíč a může vytvořit stejnou tabulku klíčů a poté dešifrovat všechny zprávy vytvořené pomocí tohoto klíče. [27]

3.2.2.1 Historie

Navzdory tomu, že ji vymyslel Charles Wheatstone, je tato šifra známá jako Playfairova. Pojmenovaná je po baronovi Lyonu Playfairovi, který prosazoval její používání. První zaznamenaný popis Playfairovy šifry je dokument podepsaný Wheatstonem, který pochází z 26. března 1854. [26] Z taktických důvodů ji Britové používali ve druhé búrské válce a první světové válce a také byla užívána Australany a Němci ve druhé světové válce. [26]

3.2.2.2 Bezpečnost

útok hrubou silou spočívá ve zkoumání četnosti výskytu bigramů a získávání klíče z nich při znalosti četnosti výskytu bigramů v předpokládaném jazyce zprávy. V angličtině je možné hledat dvojice bigramů s navzájem přehozenými písmeny, protože existuje mnoho slov začínajících RE a končících ER, stejně tak DE a ED, např. restricter nebo defeated. Po identifikování takových slov je snazší sestavit předpokládaný text a odhalit tak šifrový klíč.
Gradientní algoritmus, spočívající v náhodných záměnách např. nějakého bigramu za jiný bigram a následném zkoumání, je-li nový text smysluplnější. Ručně by se jednalo o zdlouhavou práci, nicméně počítačové systémy mohou touto metodou prolomit i poměrně krátké texty.
Playfairovu šifru odlišuje od jiných podobných šifer skutečnost, že se v žádném bigramu nevyskytují dvě stejná písmena. Pokud dostatečně dlouhý zašifrovaný text neobsahuje žádnou dvojici stejných písmen, je pravděpodobně zakódován touto šifrou.

3.2.2.3 Příkladový kód

```
public class PlayfairCipher{
//  Přetečení přes
private static int Mod(int a, int b)
{
        return (a % b + b) % b;
}
//  Vyhledání všech výskyv znaků v klíči
private static List<int> FindAllOccurrences(string str, char value)
{
List<int> indexes = new List<int>();

int index = 0;
        while ((index = str.IndexOf(value, index)) != -1)
        indexes.Add(index++);

        return indexes;
        }

//  Odebrání všech duplicitních výskytů

private static string RemoveAllDuplicates(string str, List<int> indexes)

{
string retVal = str;

for (int i = indexes.Count - 1; i >= 1; i--)
        retVal = retVal.Remove(indexes[i], 1);

return retVal;
}
//  Vytvoření klíče šifry
public static char[,] GenerateKeySquare(string key)
{
char[,] keySquare = new char[5, 5];
string defaultKeySquare = "ABCDEFGHIKLMNOPQRSTUVWXYZ";
string tempKey = key.ToUpper();

tempKey = tempKey.Replace("J", "");
tempKey += defaultKeySquare;

for (int i = 0; i < 25; ++i)
{
        List<int> indexes = FindAllOccurrences(tempKey, defaultKeySquare[i]);
        tempKey = RemoveAllDuplicates(tempKey, indexes);
}
```

```
tempKey = tempKey.Substring(0, 25);

for (int i = 0; i < 25; ++i)
            keySquare[(i / 5), (i % 5)] = tempKey[i];



return keySquare;
}
//  C# dvoudimenzionální pole není kompatibilní s Json encoderem, proto pole polí
public static char[][] ToJsonSquare(char[,] keySquare){
char[][] square = new char[5][];
for(int i = 0; i<keySquare.GetLength(0);i++){
square[i] = new char[5]
{keySquare[i,0],keySquare[i,1],keySquare[i,2],keySquare[i,3],keySquare[i,4]};
}
return square;
}
//  Vyhledání pozice prvku v dourozměrném poli
private static void GetPosition(ref char[,] keySquare, char ch, ref int row, ref int col)
{
if (ch == 'J')
GetPosition(ref keySquare, 'I', ref row, ref col);

for (int i = 0; i < 5; ++i)
            for (int j = 0; j < 5; ++j)
            if (keySquare[i, j] == ch)
            {
            row = i;
col = j;
}
}
//  Odebrání všech speciálních znaků
private static string RemoveOtherChars(string input)
{
string output = input;

for (int i = 0; i < output.Length; ++i)
            if (!char.IsLetter(output[i]))
            output = output.Remove(i, 1);
return output;
}
//  Upravení na velké a malé písmena
private static string AdjustOutput(string input, string output)
{
StringBuilder retVal = new StringBuilder(output);
```

```
for (int i = 0; i < input.Length; ++i)
{
        if (!char.IsLetter(input[i]))
        retVal = retVal.Insert(i, input[i].ToString());

        if (char.IsLower(input[i]))
        retVal[i] = char.ToLower(retVal[i]);
}
return retVal.ToString();
}
private static string Cipher(string input, char[,] keySquare, bool encipher)
{
string retVal = string.Empty;
string tempInput = RemoveOtherChars(input);
int e = encipher ? 1 : -1;
        if ((tempInput.Length % 2) != 0)
        tempInput += "X";

char[] fix = new char[tempInput.Length];
fix = tempInput.ToCharArray();
for(int i = 1; i < tempInput.Length;i+=2){
        if(tempInput[i-1] == tempInput[i]){
        fix[i] = 'X';
        continue;
        }
fix[i] = tempInput[i];
}
tempInput = new string(fix);
for (int i = 0; i < tempInput.Length; i += 2)
{
        int row1 = 0;
        int col1 = 0;
        int row2 = 0;
        int col2 = 0;

        int[] point1 = new int[2];
        int[] point2 = new int[2];
        int[] point3 = new int[2];
        int[] point4 = new int[2];
        GetPosition(ref keySquare, char.ToUpper(tempInput[i]), ref row1, ref col1);
        point3[0] = row1;
        point3[1] = col1;
        GetPosition(ref keySquare, char.ToUpper(tempInput[i + 1]), ref row2, ref col2);
        point4[0] = row2;
        point4[1] = col2;
        // Stejný řádek a sloupec
        if (row1 == row2 && col1 == col2)
        {
```

```
            point1[0] = Mod((row1 + e), 5);
            point1[1] = Mod((col1 + e), 5);
            point2[0] = Mod((row1 + e), 5);
            point2[1] = Mod((row1 + e), 5);
            }
            // Stejný řádek
            else if (row1 == row2)
            {
            point1[0] = row1;
            point1[1] = Mod((col1 + e), 5);
            point2[0] = row1;
            point2[1] = Mod((col2 + e), 5);
            }
            // Stejný sloupec
            else if(col1 == col2)
            {
            point1[0] = Mod((row1 + e), 5);
            point1[1] = col1;
            point2[0] = Mod((row2 + e),5);
            point2[1] = col1;
            }
            else
            {
            // Čtverec
            point1[0] = row1;
            point1[1] = col2;
            point2[0] = row2;
            point2[1] = col1;
            }
            / / Čtverec přidání k výsledku
            retVal += new string(new char[] { keySquare[point1[0], point1[1]], keySquare[point2[0],
point2[1]] });
}
/ / Upravení na velké a malé písmena
retVal = AdjustOutput(input, retVal);
return retVal;
}
public static string Encipher(string input, char[,] key)
{
return Cipher(input, key, true);
}
public static string Decipher(string input, char[,] key)
{
return Cipher(input, key, false);
}
}
```

Code 2: Playfairova šifra

3.2.3 Homofonní šifra

je substituční šifra, ve které se jedno písmeno z otevřeného textu může zašifrovat na různá písmena v šifrovém textu, jejichž počet je přímo úměrný frekvencí písmene. [28][29] Přitom stále používá jednu šifrovací abecedu, není to polyalfabetická šifra. Smyslem homofonní šifry je, aby ve výsledném šifrovém textu každé písmeno mělo přibližně stejnou relativní četnost, konkrétně 1 %. [28]

3.2.3.1 Historie

Jedním z prvních uživatelů homofonní šifry byl údajně vévoda Simeone de Crema z italské Mantovy na počátku 15. století. [32] Začala se hojněji využívat až v 17. stol. v Evropě.

3.2.3.2 Bezpečnost

K rozluštění lze použít frekvenční analýza pro dvojice písmen. Angličtina má časté kombinace: ee, th, qu. Přitom písmeno q (výskyt 0,1%) je následováno jedině. písmenem u (výskyt 2,8%). V šifrovém textu bude jediný znak pro q následován jedním ze tří znaků pro u. [28]

3.2.3.3 příkladový kód

```
class HomoCipher
{
public static string Encipher(string ready,Dictionary<char,string[]> key){
//Pomocné počítání
Dictionary<char,int> count_per_char = new Dictionary<char,int>();
foreach(var letter in key)
{
        count_per_char.Add(letter.Key,0);
}
// Posouvání kódu za charakter o jeden při vytváření šifrovaného textu
StringBuilder sb = new StringBuilder();
foreach(char letter in ready){
        sb.Append(key[letter][count_per_char[letter]]);
        foreach(var lkey in key){
if(lkey.Key == letter){
break;
        }
        }
        count_per_char[letter] += 1;
}
return sb.ToString();
}
public static string Decipher(string ready,Dictionary<char,string[]> key){
//získání prvního prvku ke zjištění velikosti
var e = key.GetEnumerator();
e.MoveNext();
var anElement = e.Current;
//zjištění velikosti
int digits = anElement.Value[0].Length;
StringBuilder sb = new StringBuilder();
//Jednotlivý  kód pro písmeno
string code = "";
//procházení zašifrovaného textu
for (int i = 0; i < ready.Length; i++)
{
        //přidávání k jednotlivému kódu, pokud je kompletní zjistíme , jaký znak je za ním
        code += ready[i];
        if (i % digits == 0){
        foreach(var pair in key){
        int check = Array.IndexOf(pair.Value,code);
        if(check == -1){
        continue;
        }else{
        code = "";
        }
```

```
sb.Append(pair.Key);
}
}
}
return sb.ToString();
}
public static Dictionary<char,string[]> CreateKey(string ready){
//získání frekvence znaků
Dictionary<char,int> frequency = prCharWithFreq(ready);
// List kódů za jeden znak
List<String> code = new List<String>();
// Zjisté velikost kódu za znak
int digits = countDigit(ready.Length);
string fmt = new String('0', digits);

// kódy musí mít stejnou délku tedy 000# formát
int upto = IntPow(10, digits);

// Vytváření kódů, aby byly náhodné jsou až po vyšší hranici nejen po velikost textu
for(int i = 0; i < upto; i++){
        code.Add(i.ToString(fmt));
}
//  Připravování klíče
Dictionary<char,string[]> key = new Dictionary<char,string[]>();

// Náhodné vybíraní z kódů za znak
Random rnd = new Random();
foreach(var pair in frequency)
{
        // Vytváření pole kódů za znak
        string[] c_values = new string[pair.Value];
        for(int i = 0; i < pair.Value; i++){
        int rand = rnd.Next(0, code.Count());
        c_values[i] = code[rand];
        // Odebrání z listu, aby se hodnoty neopakovaly
        code.RemoveAt(rand);
        }
//Přidání do klíče
key.Add(pair.Key,c_values);
}
return key;
}
// Mocnina pro integer
static int IntPow(int x, int pow)
{
int ret = 1;
while ( pow != 0 )
{
```

```
            if ( (pow & 1) == 1 )
            ret = ret * x;
            x = x * x;
            pow >>= 1;
}
return ret;
}
// Spočítá počet míst v čísle
static int countDigit(int n)
{
int count = 0;
while (n != 0) {
            n = n / 10;
            ++count;
}
return count;
}
// Uloží všechny písmena a jejich frekvence
static Dictionary<char,int> prCharWithFreq(string s)
{
Dictionary<char,int> d = new Dictionary<char,int>();
foreach(char i in s)
{
            if(d.ContainsKey(i))
            {
            d[i]++;
            }
            else
            {
            d[i]=1;
            }
}
return d;
}
// odebrání speciálních charakterů, zanechá jen písmena
public static string RemoveSpecialCharacters(string str) {
StringBuilder sb = new StringBuilder();
foreach (char c in str) {
if ((c >= 'a' && c <= 'z')) {
            sb.Append(c);
}
}
return sb.ToString();
}
```

Code 3: Homofonní šifra

4 Závěr

Cílem maturitního projektu bylo vytvořit sadu animací vysvětlujících historické kryptografické algoritmy. V teoretické části práce představit možnosti realizace animací v různých grafických nástrojích a na základě definovaných parametrů vybrat jeden z popsaných nástrojů pro realizaci. Výstupem práce je aplikace, která generuje animace na základě zadaného vstupu, což umožňuje uživateli (studentovi) prohloubit chápání daného problému. Aplikaci je možno dále rozšiřovat o další.
Po důkladném průzkumu problematiky a návrhu implementace jsem vytvořil konkrétní řešení.

5 Seznam příloh

6 CITACE

[1] *Vulkan Guide* [online]. [cit. 2023-03-16]. Dostupné z: https://vulkan-tutorial.com

[2] *What is Vulkan* [online]. [cit. 2023-03-16]. Dostupné z: https://developer.nvidia.com/vulkan

[3] *Vulkan [online]. [cit. 2023-03-15]. Dostupné z: https://cs.wikipedia.org/wiki/Vulkan*

[4] *Python Tutorial [online]. [cit. 2023-03-16]. Dostupné z: https://docs.python.org/3/tutorial/*

[5] *What is Python* [online]. [cit. 2023-03-16]. Dostupné z: https://www.python.org/doc/essays/blurb/

[6] *Why Python* [online]. [cit. 2023-03-16]. Dostupné z: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python

[7] *Tkinter* [online]. [cit. 2023-03-16]. Dostupné z: https://cs.wikipedia.org/wiki/Tkinter

[8] *Tkinter Python interface to Tcl/Tk* [online]. [cit. 2023-03-16]. Dostupné z: https://docs.python.org/3/library/tkinter.html

[9] *Turtle graphics* [online]. [cit. 2023-03-16]. Dostupné z: https://docs.python.org/3/library/turtle.html

[10] *The Beginner's Guide to Python Turtle* [online]. [cit. 2023-03-16]. Dostupné z: https://realpython.com/beginners-guide-python-turtle/

[11] *Why Rust* [online]. [cit. 2023-03-16]. Dostupné z: https://www.rust-lang.org/

[12] *How Rust went from a side project to the world's most-loved programming language* [online]. [cit. 2023-03-16]. Dostupné z: https://www.technologyreview.com/2023/02/14/1067869/rust-worlds-fastest-growing-programming-language/

[13] *A cross-platform GUI library for Rust, inspired by Elm* [online]. [cit. 2023-03-16]. Dostupné z: https://github.com/iced-rs/iced

[14] *Introduction to iced* [online]. [cit. 2023-03-16]. Dostupné z: https://book.iced.rs/

[15] *About egui* [online]. [cit. 2023-03-16]. Dostupné z: https://www.egui.rs/

[16] *The state of Rust GUI libraries* [online]. [cit. 2023-03-16]. Dostupné z: https://blog.logrocket.com/state-of-rust-gui-libraries/

[17] *Flutter (software)* [online]. [cit. 2023-03-16]. Dostupné z: https://en.wikipedia.org/wiki/Flutter_(software)

[18] *Why Flutter is the most popular cross-platform mobile SDK* [online]. [cit.

2023-03-16]. Dostupné z: https://stackoverflow.blog/2022/02/21/why-flutter-is-the-most-popular-cross-platform-mobile-sdk/

[19] *Introduction to SolidJS* [online]. [cit. 2023-03-16]. Dostupné z: https://www.loginradius.com/blog/engineering/guest-post/introduction-to-solidjs/

[20] *React (webový framework)* [online]. [cit. 2023-03-16]. Dostupné z: https://cs.wikipedia.org/wiki/React_(webov%C3%BD_framework)

[21] *SolidJS vs. React: The Go-to Guide* [online]. [cit. 2023-03-03]. Dostupné z: https://www.toptal.com/react/solidjs-vs-react

[22] *What is ASP.NET* [online]. [cit. 2023-03-06]. Dostupné z: https://dotnet.microsoft.com/en-us/learn/aspnet/what-is-aspnet

[23] *Node.js vs. ASP.NET - Which is a Better Option for Enterprise Application Development?* [online]. [cit. 2023-03-16]. Dostupné z: https://www.turing.com/kb/node-js-vs-asp-net

[24] *Caesarova šifra – Wikipedie* [online]. [cit. 2023-03-03]. Dostupné z: https://cs.wikipedia.org/wiki/Caesarova_%C5%A1ifra

[25] *Caesar Cipher in Cryptography* [online]. [cit. 2023-03-06]. Dostupné z: https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/

[26] *Playfairova šifra – Wikipedie* [online]. [cit. 2023-03-03]. Dostupné z: https://cs.wikipedia.org/wiki/Playfairova_%C5%A1ifra

[27] *Playfair Cipher with Examples* [online]. [cit. 2023-03-06]. Dostupné z: https://www.geeksforgeeks.org/playfair-cipher-with-examples/

[28] *Homofonní šifra – Wikipedie* [online]. [cit. 2023-03-03]. Dostupné z: https://cs.wikipedia.org/wiki/Homofonn%C3%AD_%C5%A1ifra

[29] *Homofonní šifra* [online]. [cit. 2023-03-06]. Dostupné z: https://czwiki.cz/Lexikon/Homofonn%C3%AD_%C5%A1ifra

[30] *Frekvenční analýza – Wikipedie* [online]. [cit. 2023-03-03]. Dostupné z: https://cs.wikipedia.org/wiki/Frekven%C4%8Dn%C3%AD_anal%C3%BDza

[31] *Frequency analysis with examples* [online]. [cit. 2023-03-06]. Dostupné z: https://www.geeksforgeeks.org/program-to-perform-a-letter-frequency-attack-on-a-monoalphabetic-substitution-cipher/

[32] GOLLOVÁ, Alena. Dějiny kryptografie [online]. [cit. 2023-03-30]. Dostupné z: https://math.fel.cvut.cz/en/people/gollova/mkr/mkr0.pdf