LUZERN

KANTON LUZERN

Programmieren eines Escape Room Videospiels über kryptografische Methoden

Maturaarbeit

Medea Emch, G19f

29. September 2022

Betreuung: Adrian Häfliger, Fachschaft Informatik

Kantonsschule Alpenquai Luzern | ksalpenquai.lu.ch

Abstract Gegenstand der vorliegenden Maturaarbeit ist der Prozess des Programmierens eines Videospiels und die Recherche und Aufarbeitung der darin vorgestellten kryptografischen Methoden.

Dazu wurden die kryptografischen Methoden analysiert, in einen Text aufgearbeitet und anschliessend mit Python programmiert. Zusätzlich wurde ebenfalls mit Python ein Escape Room Videospiel programmiert, in dem man einige der kryptografischen Methoden anwenden muss, um aus den Räumen zu entkommen.

Inhaltsverzeichnis

1 Vorwort und Danksagung

Anfangs fiel es mir schwer Ideen für eine Maturaarbeit zu finden. Es war mir klar, dass es ein Thema im Bereich von Mathematik und Informatik sein sollte, aber ich konnte mir kaum vorstellen, wie so ein Thema oder eine Leitfrage aussehen sollte.

Da ich Ende Schuljahr 2020/2021 als Mathematik-Projekt ein kleines Jump-and-Run Game mit Python programmiert habe, war meine erste Idee erneut ein Game zu programmieren, aber in einem anderen Genre. Genauere Ideen, was für ein Game es werden könnte, hatte ich nicht.

Schlussendlich verfasste ich eine Liste mit allen Themen, die mich interessierten und ging damit an eine erste Besprechung mit meinem potenziellen Betreuer, Herrn Adrian Häfliger, der mich darauf hinwies, dass ich Punkte meiner Liste auch kombinieren könnte. Daraus und aus meiner Liebe zu Escape Rooms kam mir die Idee, ein Escape Room Game über Kryptografie zu programmieren, da Kryptografie ein Thema meiner Liste war.

Ich möchte mich zuallererst bei Herrn Adrian Häfliger bedanken, der diese Maturaarbeit sehr unkompliziert betreut hat und immer zur Verfügung stand, wenn irgendwelche Fragen oder Unsicherheiten auftauchten. Weiter bedanke ich mich bei meinen Eltern, die mich in den Sommerferien und auch danach auf Trab gehalten haben, speziell meinem Vater, der immer bereit war zu helfen, besonders beim Programmieren. Er sorgte dafür, dass ich alle meine Dateien auf GitHub speicherte, wofür ich im Nachhinein unendlich dankbar bin. Auch meinen Freunden, die meine zum Teil endlosen Ausführungen über Kryptografie und Programmieren ertragen haben, gebührt Dank.

2 Einleitung

Kryptografie ist besonders faszinierend, da jeder täglich davon Gebrauch macht und der Begriff trotzdem kaum jemandem etwas sagt. Aber gäbe es die Kryptografie nicht, wäre es unmöglich, einen sicheren elektronischen Zahlungsverkehr aufzubauen, vertrauliche Daten auf einem Computer zu speichern oder das Internet in einem Ausmass zu nutzen, wie es heute für viele Menschen Gewohnheit ist.

Doch die Kryptografie gibt es nicht erst seit dem Internet und den digitalen Medien. Ursprünglich kam sie aus dem Kriegswesen der Antike. Dabei war die Hauptanwendung, Befehle sicher an Verbündete übermitteln zu können und dabei die Gefahr, dass etwaige Feinde die Nachricht abfangen und mitlesen, zu minimieren. Verschlüsselungsverfahren und Geheimschriften wurden auch häufig für andere vertrauliche Informationen benutzt, wie etwa Rezepte für Heilmittel und Liebesbriefe. Kryptografie wurde in vielen Kriegen verwendet und hat auch in den meisten einen massgeblichen Unterschied gemacht, wie etwa im zweiten Weltkrieg, als es Alan Turing, einem britischen Mathematiker und Kryptanalyst gelang, das Enigma-Verschlüsselungssystem der Deutschen zu knacken. Dabei hat er unter anderem auch die Grundbausteine für die moderne Informatik und moderne Computer gelegt.

Heutzutage findet Kryptografie vor allem in der digitalen Welt Gebrauch, wobei es nicht mehr nur darum geht, Nachrichten in ein unleserliches Format zu verschlüsseln, sondern auch um Datenschutz und Datensicherheit. Dabei gibt es folgende Hauptziele:

- Es sollen nur dazu berechtigte Personen in der Lage sein, Daten zu lesen oder sonstige Informationen über ihren Inhalt zu erlangen.
- Es soll nachweisbar sein, dass die Daten vollständig und unverändert sind.
- Es soll eindeutig und auch gegenüber Dritten nachweisbar sein, wer der Urheber der Daten ist und diese Urheberschaft soll unbestreitbar sein.

Das Ziel dieser Maturaarbeit ist, einige der simpleren Verschlüsselungsmethoden aufzuarbeiten und in Form eines Spiels verständlich zu machen. Für die Programmierung dieses Spiels wird Python (Version 3.9.5), eine als übersichtlich und einfach zu erlernend geltende Programmiersprache verwendet. Weiter wird die Pygame-Bibliothek (Version 2.1.2) genutzt, eine Erweiterung für Python spezifisch für das Programmieren von Videospielen, die viele der sonst anstrengenderen Teile des Spiele-Programmierens, wie zum Beispiel die Grafik, einfacher macht. Zuletzt wird Visual Studio Code, eine integrierte Entwicklerumbgebung von Microsoft, benutzt, die unter anderem Python unterstützt.

3 Behandelte kryptografische Methoden

3.1 Geheimtinte

Die einfachste Art, einen geheimen Text zu vermitteln ist, wenn er gar nicht erst gesehen werden kann. Diese Art von «Verschlüsselung» gehört nicht zum Themenbereich der Kryptografie, sondern zu dem der Steganografie. Diese ist ein benachbartes Feld der Kryptografie, da sie sich ebenfalls mit der Sicherheit von Daten und Nachrichten beschäftigt. Statt wie in der Kryptografie die Nachricht selbst zu verschlüsseln, beschäftigt sich die Steganografie mit dem Verstecken der Daten, Nachrichten oder Kanäle, auf denen sie übertragen werden, sodass sie gar nicht erst in falsche Hände gelangen können.

3.1.1 Funktionsweise

Dies geschieht zum Beispiel mit Geheimtinten, verschiedenen Chemikalien, die unter Normalbedingungen farblos sind und entweder durch Hitze, UV-Licht oder chemische Behandlungen sichtbar werden oder indem die Nachricht in einer anderen nicht geheimen Nachricht versteckt wird. Damit ein «leeres» Blatt Papier nicht verdächtig erscheint, wird oft eine unverfängliche Nachricht darauf geschrieben. Dabei wird die Geheimnachricht entweder zwischen die Zeilen oder auf den Rand des Blattes geschrieben oder es werden mit der Geheimtinte lediglich bestimmte Wörter oder Buchstaben markiert, die anschliessend zusammen die Geheimnachricht ergeben.

3.1.2 Geschichte

Einer der frühesten Autoren, der unsichtbare Tinte erwähnt, war Aineias Taktikos, ein griechischer Stratege und Militärschriftsteller aus dem 4. Jahrhundert v. Chr. Im alten Griechenland und im römischen Reich wurden Geheimtinten noch zu Kriegszwecken genutzt, da es damals noch keine elektronische Datenübertragung gab und Nachrichten zumeist niedergeschrieben wurden, wofür sich Geheimtinte sehr eignete. Später, im 17. bis 19. Jahrhundert erlebten Geheimtinten einen Popularitätsschub, um geheime Liebesbriefe zu schreiben. Heutzutage werden Geheimtinten fast ausschliesslich von Kindern für «geheime» Nachrichten genutzt. In der digitalen Welt finden Geheimtinten keine Anwendung, da sie sich nur für physisch niedergeschriebene Nachrichten eignen.

Die frühesten Formen von unsichtbaren Tinten waren Milch, Urin und Fruchtsäfte. Über die angetrocknete Milch blies man Kohlestaub oder Asche, welche am fettigen Rückstand der getrockneten Milch kleben blieben. Milch, Urin und Fruchtsäfte werden dunkler bei Erwärmung, da deren enthaltene Kohlenhydrate (Kohlenstoff) verkohlen, wie man es

auch zum Beispiel beim Verbrennen von Holz beobachten kann. Später fand man auch in der Chemie mehr Möglichkeiten für Geheimtinten. Diese sind zumeist Substanzen, die auf bestimmte andere Chemikalien reagieren und sich daraufhin verfärben. Ein Beispiel dafür ist Phenolphthaleinlösung, ein pH-Wert-Indikator, der bei einem pH-Wert von 0 bis 8,2 farblos ist. Beim Bestreichen mit einer stärkeren Base wie zum Beispiel einer Ammoniaklösung, wird das Phenolphthalein pink.

Tabelle 1: Die Generation des Geheimalphabets aus dem Klartextalphabet

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Tabelle 2: Die Chiffrierung nach Caesar aus dem Anwendungsbeispiel

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↓	↓	↓	...																					...	↓
K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J

3.2 Caesar-Verschlüsselung

Caesar ist eine der simpelsten Verschlüsselungen und wird heute fast ausschliesslich benutzt, um das Prinzip der Kryptografie einfach zu erklären, da sie für heutige Verwendungen viel zu unsicher ist.

Caesar ist eine monoalphabetische Verschlüsselung, bei der ein festes Geheimalphabet benutzt wird und ein Zeichen oder eine Zeichengruppe im Klartext immer dem gleichen Zeichen oder der gleichen Zeichengruppe im Geheimtext entspricht und umgekehrt. Solche Verfahren sind einfach zu benutzen, aber nicht sehr sicher.

Für die Caesar-Verschlüsselung wird ein Klartextalphabet und ein Geheimalphabet benutzt, wobei jeder Buchstabe im Klartextalphabet jeweils dem Buchstaben an derselben Stelle des Geheimalphabets entspricht und mit diesem ersetzt wird.

3.2.1 Generation des Klartext- und Geheimalphabets

Das verschlüsselte Alphabet erhält man, indem man die Zeichen des lateinischen Alphabets um eine bestimmte Anzahl Stellen verschiebt, wobei der Anfang des Klartextalphabets zyklisch am Ende des Alphabets angefügt wird. Um anzugeben, um wie viel das Alphabet verschoben wurde, wird entweder die Anzahl der Stellen oder der Schlüsselbuchstabe (der Buchstabe, durch den A ersetzt wurde) angegeben.

Für die Entschlüsselung einer mit Caesar verschlüsselten Nachricht muss aus dem Geheimalphabet wieder das Klartextalphabet gewonnen werden, um den Prozess rückwärts anwenden zu können. Dafür wird das Geheimalphabet um die gleiche Anzahl Stellen zurückverschoben.

Tabelle 3: Verschlüsselung des Wortes "Beispiel"nach Caesar (Schlüsselzahl: 10)

Klartext:	B	E	I	S	P	I	E	L
Geheimtext:	L	O	S	C	Z	S	O	V

3.2.2 Anwendungsbeispiel

Aus dem Wort «BEISPIEL» wird somit «LOSCZSOV», wenn es mit der Schlüsselzahl 10 verschlüsselt wird.

Um den Geheimtext wieder zu entschlüsseln wird der ganze Vorgang rückwärts angewandt. Der Schlüssel (in diesem Fall K) wird dabei von A ersetzt.

3.2.3 Mathematische Darstellung

Die Caesar-Verschlüsselung kann mathematisch dargestellt werden, indem man jedem der 26 Buchstaben eine Zahl zuordnet (A = 0, B = 1, ..., Z = 25). Mit diesen Zahlen kann man die Caesar-Verschlüsselung als ganz einfache Addition darstellen. Dazu wird zum Wert jedes Klartextbuchstabens K_i einfach der Wert des Schlüsselbuchstabens S addiert.

Da es aber Fälle gibt, in denen das Resultat grösser als 25 ist und es keinen Buchstaben mit einem so hohen Wert gibt, muss auf das Resultat eine Modulo-26 Rechnung angewandt werden. Dabei wird der Rest einer Division durch 26 berechnet.

Somit ist die Caesar-Verschlüsselung mathematisch dargestellt als:

Die dazugehörende Entschlüsselung eines Geheimtextbuchstabens Gi entspricht dann:

3.2.4 Sicherheit

Da die Caesar-Verschlüsselung eine monoalphabetische Verschlüsselung ist, das heisst, jeder Klartextbuchstabe im Klartextalphabet genau einem Geheimbuchstaben im Geheimalphabet entspricht, kann sie durch Statistik sehr leicht geknackt werden. Jede Sprache hat eine charakteristische Verteilung der Buchstaben, die leicht in einem Graph aufgezeichnet werden kann. In der deutschen Sprache sieht diese Verteilung folgendermassen aus:

Wenn also das ganze Alphabet um beispielsweise 10 Stellen verschoben wird, sieht die Verteilung folgendermassen aus:

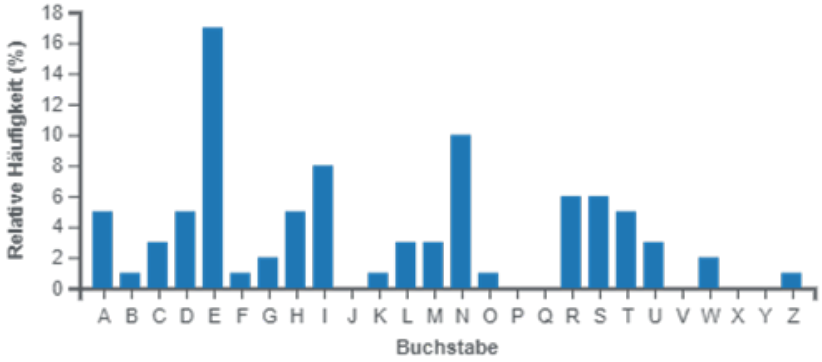

Abbildung 1: Verteilung der Buchstaben in der deutschen Sprache [**img:citekey**]

Abbildung 2: Bild mit Caesar, 10 Zeichen verschoben, Buchstabenverteilung [**img:citekey**]

Da der Verlauf des Graphen immer noch derselbe ist und man weiss, dass E der am häufigsten verwendete Buchstabe der deutschen Sprache ist, kann man daraus schliessen, dass das O des Geheimalphabets dem E des Klartextalphabets entspricht. So kann man die Verschiebung berechnen und daraus das Klartextalphabet ableiten.

Da die Buchstabenverteilung jedoch erst in genügend langen Texten ausreichend aussagekräftig ist, sollten einzelne Wörter und kurze Sätze in dieser Hinsicht noch einigermassen sicher sein.

Allerdings ist eine weitere Schwäche der Caesar-Verschlüsselung die Limitierung durch die nur 25 möglichen Schlüssel. Man erhält also, auch wenn man bloss herumprobiert, spätestens nach 25 Versuchen den Klartext. Vor dieser Strategie sind dann auch kurze Sätze und einzelne Wörter nicht sicher.

Im Englischen gibt es zusätzlich noch das Problem, dass es nur zwei Möglichkeiten gibt für Wörter mit nur einem Buchstaben («I» = ich und «a» = ein), was das Knacken noch zusätzlich beschleunigt, besonders da beides eher häufige Wörter sind. Dies kann

aber umgangen werden, indem alle Leerzeichen des Geheimtextes bei der Übermittlung weggelassen werden, wodurch man einzelne Wörter nicht mehr erkennen kann.

3.2.5 Geschichte

Die Caesar-Verschlüsselung wurde nach Gaius Julius Caesar benannt, der laut Überlieferung durch den römischen Schriftsteller Sueton eine solche Verschlüsselung mit einer Verschiebung von drei Stellen verwendet hat.

Abbildung 3: Ein Exemplar der Caesar-Scheibe [**img:citekey**]

Im 15. Jahrhundert erfand Leon Battista Alberti, ein italienischer Schriftsteller, Architekt und Mathematiker, die Chiffrierscheibe, ein Werkzeug, welches die Nutzung der Caesar-Verschlüsselung vereinfacht. Eine solche Chiffrierscheibe besteht aus zwei runden, konzentrischen Scheiben, die so aneinander angebracht sind, dass die kleinere Scheibe sich auf der grösseren drehen kann. Entlang dem Rand der einen Scheibe sind alle Zeichen des Klartextalphabets angeschrieben und entlang dem Rand der anderen Scheibe alle Zeichen des Geheimalphabets. Für jede der verschiedenen Caesar-Verschlüsselungen müssen einfach die Scheiben in einen bestimmten Winkel zueinander gedreht werden. Dann können direkt die zueinander gehörenden Buchstaben abgelesen werden.

Tabelle 4: Tabula Recta der Atbasch-Verschlüsselung

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A

Tabelle 5: Tabula Recta der ROT13-Verschlüsselung

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕	↕
N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M

3.2.6 Varianten

Atbasch Atbasch ist eine ursprünglich auf dem hebräischen Alphabet basierende Variante der Caesar-Verschlüsselung, die auch als umgekehrte Caesar-Verschlüsselung bezeichnet wird. Statt die Buchstaben um eine bestimmte Anzahl Stellen zu verschieben, entspricht das Geheimalphabet lediglich dem Klartextalphabet, aber rückwärts. So wird A zu Z, B zu Y, und so weiter:

Der Name Atbasch leitet sich dabei von den ersten zwei Buchstabenpaaren des hebräischen Alphabets ab, die einander ersetzen (**A**leph und **T**aw und **B**eth und **Sch**in). Speziell an Atbasch ist, dass zum Entschlüsseln der gleiche Prozess benutzt werden kann wie zum Verschlüsseln, da die Buchstaben symmetrisch ausgetauscht werden.

ROT13 ROT13 ist eine weitere Variante der Caesar-Verschlüsselung, die den gleichen Prozess zum Verschlüsseln und Entschlüsseln benutzt. Hier sind die Buchstaben zwar wie in der normalen Caesar-Verschlüsselung verschoben, aber genau um ein halbes Alphabet, also 13 Stellen. Wenn man also einen Buchstaben verschlüsselt (um 13 Stellen verschiebt) und noch einmal verschlüsselt (um weitere 13 Stellen verschiebt), hat man den Buchstaben um insgesamt 26 Stellen, also ein ganzes Alphabet verschoben, womit man wieder beim Ausgangsbuchstaben landet.

Aufgrund ihrer Einfachheit wird ROT13 oft verwendet, um beispielsweise Spoiler im Internet unleserlich zu machen, sodass man sie nicht aus Versehen lesen kann.

Tabelle 6: Tabula Recta der Vigenère-Verschlüsselung

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

3.3 Vigenère-Verschlüsselung

Anders als bei der Cäsar-Verschlüsselung wird bei der Vigenère-Verschlüsselung ein Schlüssel in Kombination mit 26 Geheimalphabeten benutzt. Dabei wird der Schlüssel so oft wiederholt, bis er die Länge der zu verschlüsselnden Nachricht deckt.

Im Grunde genommen wird für jeden Buchstaben des Klartexts eine neue Caesar-Verschlüsselung mit einer neuen Verschiebung verwendet, wobei die Verschiebung an jeder Stelle dem Buchstaben im Schlüssel an dieser Stelle entspricht.

3.3.1 Anwendungsbeispiel

Um die Vigenère-Verschlüsselung effizient von Hand nutzen zu können, wird eine Tabula Recta (lat. Quadratische Tafel) verwendet. Diese ist eine Liste aller 26 möglichen Cäsar-Geheimalphabete:

Wenn man das Wort «BEISPIEL» nun mit dem Schlüssel «KEY» verschlüsseln will, beginnt man mit der ersten Stelle und verschlüsselt den Klartextbuchstaben «B» mit dem Schlüsselbuchstaben «K». Dazu wird zuerst der Klartextbuchstabe in der obersten Zeile und der Schlüsselbuchstabe in der Spalte ganz links gesucht und markiert. Nun wird

Tabelle 7: Verschlüsselung des Wortes "Beispiel"nach Vigenère (Schlüssel: "KEY")

Klartext:	B	E	I	S	P	I	E	L
Schlüssel:	K	E	Y	K	E	Y	K	E
Geheimtext:	L	I	G	C	T	G	O	P

der Buchstabe gesucht, der in derselben Spalte wie der markierte Klartextbuchstabe und in derselben Zeile wie der markierte Schlüsselbuchstabe steht. Dies ist der verschlüsselte Buchstabe. So erhält man den Geheimtext «LIGCTGOP».

3.3.2 Mathematische Darstellung

Die Vigenère-Verschlüsselung kann ebenfalls mathematisch dargestellt werden. Der einzige Unterschied zur mathematischen Darstellung der Cäsar-Verschlüsselung liegt darin, dass für die Verschlüsselung jedes einzelnen Klartextbuchstabens K_i der entsprechende Schlüsselbuchstabe S_i benutzt wird, anstatt eines konstanten Schlüsselbuchstabens. Die mathematischen Darstellungen für die Ver- und Entschlüsselung sehen also folgendermassen aus:

3.3.3 Sicherheit

Die Vigenère-Verschlüsselung galt lange als unknackbar, doch im Jahr 1854 gelang es Charles Babbage, einem englischen Mathematiker, Philosoph und Erfinder, eine mit Vigenère verschlüsselte Nachricht zu entziffern. Seine Methode hielt er jedoch geheim. Erst 1863 wurde von Friedrich Wilhelm Kasiski, einem preussischen Infanteriemajor, eine erfolgreiche Methode veröffentlicht. Dieses Verfahren ist heute als Kasiski-Test bekannt. Die mit dem Kasiski-Test ausgenutzten Schwachpunkte der Vigenère-Verschlüsselung sind die Wiederholung des Schlüssels, und die Einschränkung der möglichen Schlüssel auf bereits existierende Wörter. Durch Analyse von im Geheimtext mehrfach auftretenden Buchstabenfolgen kann die Länge des Schlüssels bestimmt werden, woraus man dann mit Statistik und etwas Ausprobieren auf den Schlüssel und den Klartext oder zumindest Teile davon schliessen kann.

Um eine komplette Sicherheit zu gewährleisten, müsste der Schlüssel eine einzigartige Aneinanderreihung von zufälligen Buchstaben ohne jegliche Wiederholungen, mindestens so lange wie der Klartext und vollständig geheim sein. Diese Verschlüsselungstechnik ist bekannt als One Time Pad.

3.3.4 Varianten

Trithemius-Verschlüsselung Die Verschlüsselung nach Trithemius ist der Vorläufer der Vigenère-Verschlüsselung und wurde vom deutschen Autor und Mönch Johannes Trithemius im frühen 16. Jahrhundert zusammen mit der Tabula Recta erfunden. Für diese Verschlüsselung benutzt man auch die Tabula Recta aber im Vergleich zu der normalen Vigenère-Verschlüsselung keinen Schlüssel, sondern man rückt bei jedem Buchstaben eine Zeile der Tabula Recta weiter nach unten. Im Grunde genommen ist die Trithemius-Verschlüsselung also eine Vigenère-Verschlüsselung mit einem fixen Schlüssel «ABCDEFGHIJKLMNOPQRSTUVWXYZ».

Beaufort-Variante Da die Vigenère-Verschlüsselung nicht reziprok ist, das Vorgehen des Verschlüsselns also nicht dem Vorgehen des Entschlüsselns entspricht, kann man auch in die «falsche» Richtung verschlüsseln. Dazu wird auf den Klartext der Algorithmus zur Entschlüsselung nach Vigenère angewandt, um den Geheimtext zu erhalten, sodass man es nachher mit der normalen Verschlüsselungstechnik wieder entschlüsseln kann. Dies wird als Beaufort Variante bezeichnet und ist nicht zu verwechseln mit der Beaufort-Verschlüsselung.

Beaufort-Verschlüsselung Für die Beaufort-Verschlüsselung wird eine ähnliche Tabelle benutzt wie die Tabula Recta, allerdings ist darin das Alphabet rückwärts:

Beim Testen dieser Verschlüsselung fiel die Bildung von immer gleichen Buchstabentripeln auf, bei denen man immer auf den dritten Buchstaben schliessen kann, wenn zwei bekannt sind, egal wie die drei Buchstaben auf Klartext, Geheimtext und Schlüssel verteilt sind. Beispiele dieser Tripel sind AAZ, ABY, ACX etc. Dank dieser Tripel ist die Beaufort-Verschlüsselung reziprok und trotz ihrer Sicherheit relativ einfach von Hand zu benutzen, wenn man einmal die Tripel kennt. Da man die Reihenfolge der Buchstaben im Tripel in irgendeiner Reihenfolge lernen kann, könnten daraus auch gut Eselsbrücken gemacht werden.

Gronsfeld-Verschlüsselung Für diese Variante der Vigenère-Verschlüsselung werden als Schlüssel keine Buchstaben, sondern Ziffern benutzt. Da es nur 10 Ziffern gibt, gibt es also auch nur 10 Schlüsselalphabete, was diese Verschlüsselung etwas unsicherer macht als die Vigenère-Verschlüsselung.

Was ihre Sicherheit aber wieder verstärkt ist die zusätzliche Unvorhersehbarkeit, mit der zu rechnen ist, da der Schlüssel aus Ziffern besteht und deshalb kein existierendes Wort sein kann.

Tabelle 8: Tabula Recta der Beaufort-Verschlüsselung

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
B	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z
C	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y
D	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X
E	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W
F	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V
G	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U
H	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T
I	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S
J	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R
K	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q
L	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P
M	N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O
N	M	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N
O	L	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M
P	K	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L
Q	J	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K
R	I	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J
S	H	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I
T	G	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H
U	F	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G
V	E	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F
W	D	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E
X	C	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D
Y	B	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C
Z	A	Z	Y	X	W	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B

Tabelle 9: Tabula Recta der Gronsfeld-Verschlüsselung

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**Z**
0	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I

Autokey-Verschlüsselung Dies ist die eigentliche Methode, die Blaise de Vigenère, nach dem die Vigenère-Verschlüsselung benannt wurde, erfunden hat. Für die Autokey-Verschlüsselung wird ein Schlüssel verwendet, der automatisch länger ist als der Klartext. Dafür wird ein Vorschlüssel benutzt, der für die ersten paar Zeichen zum Verschlüsseln benutzt wird. Statt den Schlüssel so lange zu wiederholen, bis er die ganze Länge des Klartexts abdeckt, wird an den Schlüssel der Klartext angehängt. Hat man also einen fünfstelligen Schlüssel, wird für die Verschlüsselung der 6. Stelle des Klartextes, die erste Stelle des Klartextes als Schlüssel benutzt.

Running Key-Verschlüsselung Auch bei dieser Variante wird ein Schlüssel benutzt, der mindestens so lang ist wie der Klartext. Hier wird dafür einfach ein langer Schlüssel benutzt. Da diese aber schwieriger sind sich zu merken werden oft Passagen von Büchern oder öffentlichen Texten als Schlüssel benutzt.

4 Das Escape Room Spiel

4.1 Benutzeroberfläche

Wenn das Spiel geöffnet wird, erscheint als erstes das Hauptmenü des Spiels.

Abbildung 4: Das Hauptmenu

Im Hauptmenü gibt es zwei Knöpfe, den Startknopf (1), der das Spiel startet und den Spieler in den ersten Raum setzt und den Beenden-Knopf (2), der das Spiel stoppt und das Programm schliesst.

Das Spiel wird ausschliesslich mit der Maus gespielt, Tastatureingaben funktionieren nicht.

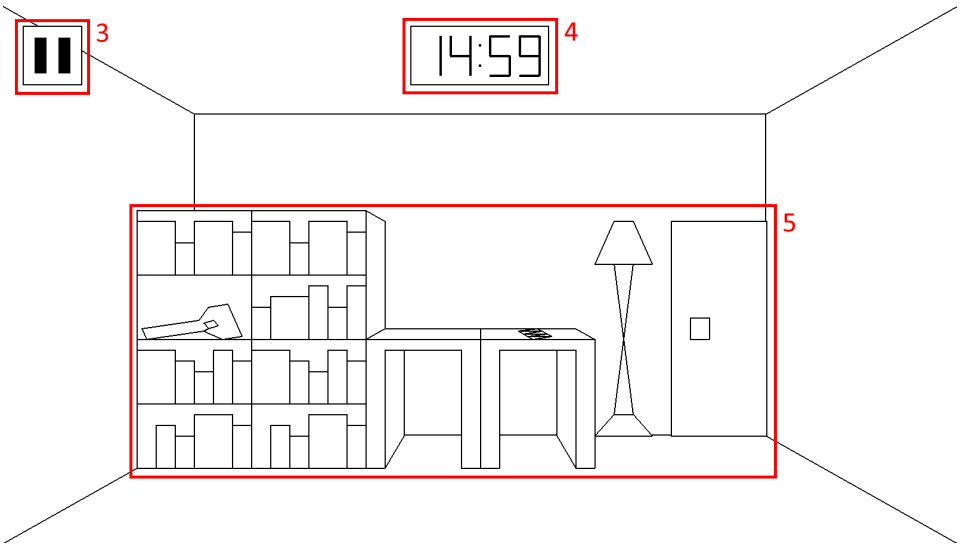

Abbildung 5: Die Hauptansicht des ersten Raumes

Wenn das Spiel gestartet wird, erscheint die Einrichtung des ersten Raumes (5), der Pausenknopf (3) und ein Timer (4). Das Ziel ist es, in allen Räumen das korrekte Passwort oder den korrekten Code einzugeben, bevor der Timer abläuft. Mit den meisten Objekten aus der Hauptansicht (5) kann interagiert werden, so kann zum Beispiel auf das Regal ganz links geklickt werden, was eine neue, genauere Ansicht des Objekts anzeigt.

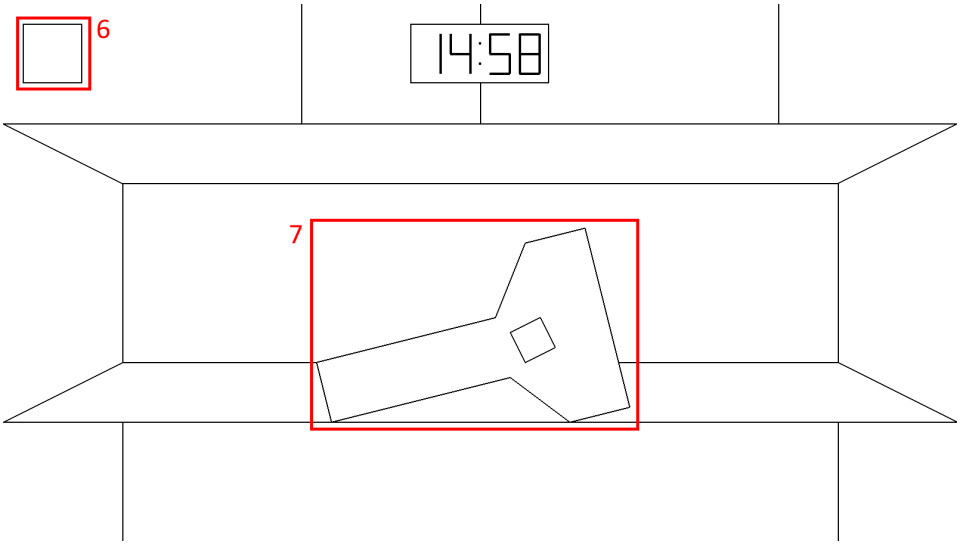

Abbildung 6: Die Detailansicht des Regals mit der UV-Taschenlampe

In den genaueren Ansichten von Objekten, wird der Pausenknopf durch einen Knopf (6) ersetzt, mit dem man zur Hauptansicht zurückwechseln kann. In den meisten Detailansichten von Objekten besteht die Möglichkeit weiter damit zu interagieren, so wie man beispielsweise hier die UV-Taschenlampe anklicken kann, wodurch sie aufgehoben wird.

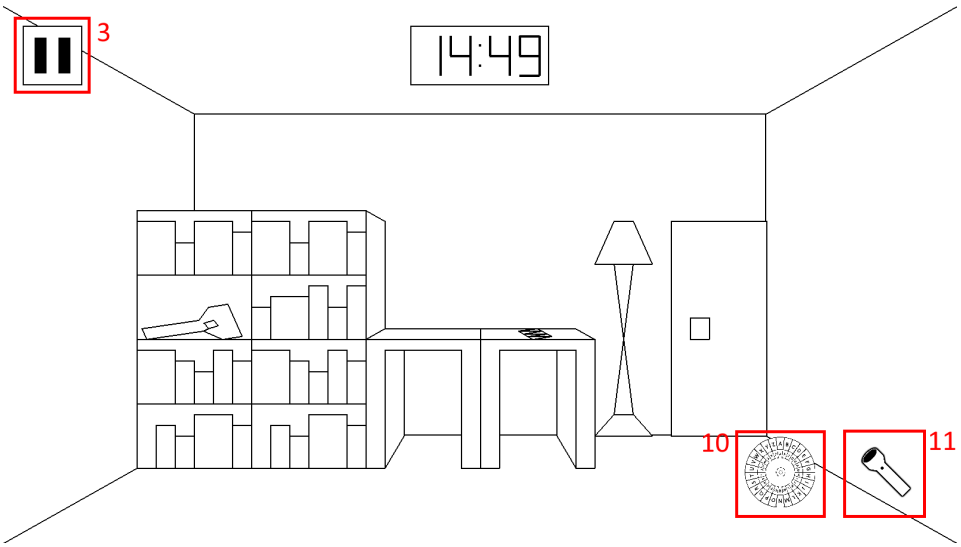

Abbildung 7: Die Hauptansicht mit aufgehobenen Werkzeugen

Aufgehobene Werkzeuge werden in der rechten unteren Bildschirmecke in Form von Knöpfen angezeigt. In der Abbildung 4 trägt der Spieler sowohl die UV-Taschenlampe (11) als auch die Chiffrierschiebe (10). Mit Klicken auf eins der Werkzeuge kann dieses benutzt werden.

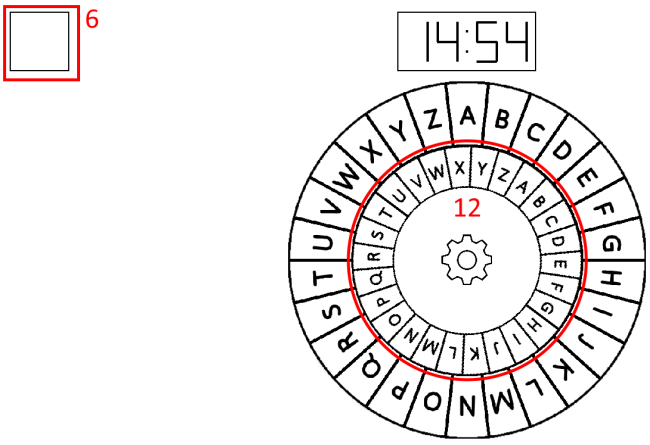

Abbildung 8: Die Chiffrierscheibe

Im Falle der Chiffrierscheibe öffnet sich eine grosse Ansicht der Chiffrierscheibe, die auch benutzt werden kann. Dazu klickt man mit der Maus auf die innere Scheibe (12) und hält sie mit der Maus fest, während man daran dreht. Auch in dieser Ansicht befindet sich ein Knopf (6) mit der zurück zur Hauptansicht geschalten werden kann.

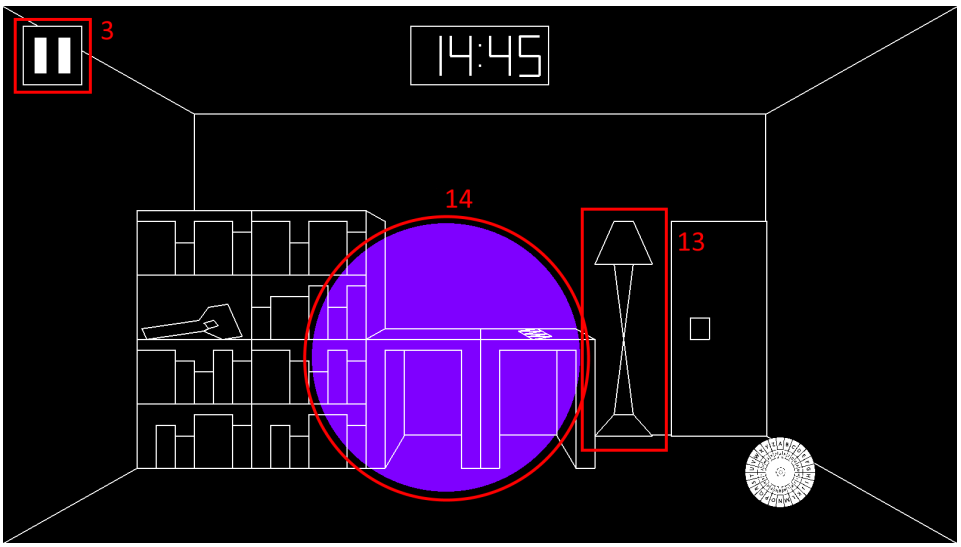

Abbildung 9: Die Benutzung der UV-Taschenlampe

Die Taschenlampe funktioniert ausschliesslich, solange das Licht im Raum ausgeschalten ist, was man jeweils durch Anklicken der Lampe (13) im Zimmer macht. Die Taschenlampe muss ebenfalls mit einem Mausklick «gepackt» werden. Solange man sie festhält, kann man den Mauszeiger bewegen und so den Lichtschein der Lampe (14) kontrollieren.

Mit dem Pausenknopf (3) von der Hauptansicht kann das Spiel, sowie der Timer pausiert werden, woraufhin das Pausenmenü angezeigt wird.

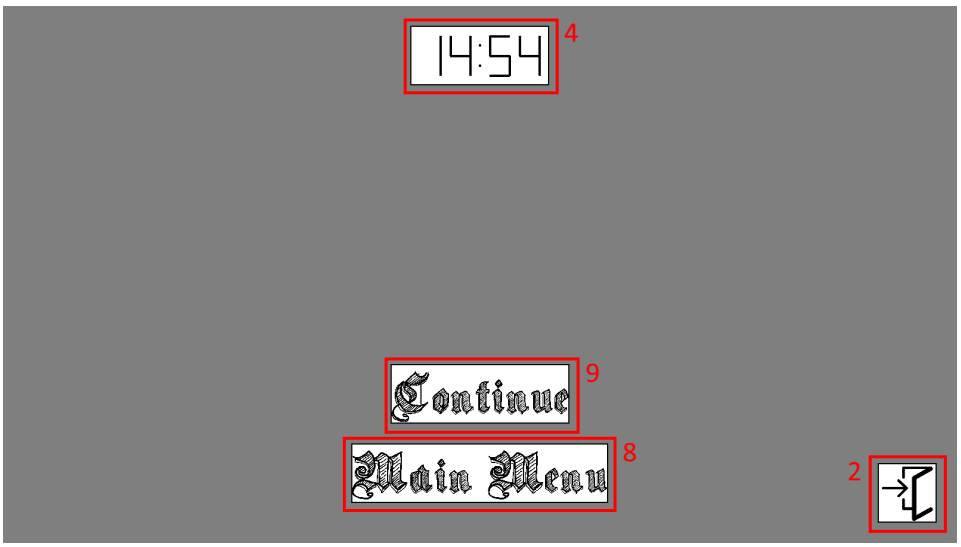

Abbildung 10: Das Pausenmenü

Der Timer (4) wird weiterhin angezeigt, ist aber eingefroren. Weiter erscheint der Knopf zum Beenden des Spiels (2) wie im Hauptmenü. Mit dem Fortsetzen-Knopf (9) kann

man ins Spiel zurückkehren, um dort weiterzuspielen wo aufgehört wurde. Dadurch wird auch der Timer fortgesetzt. Mit dem Hauptmenü-Knopf (8) kann man ins Hauptmenü zurückkehren, wo ein neues Spiel gestartet werden kann.

4.2 Das Programm

4.2.1 Vorteile und Nachteile von Python

Python ist eine vergleichsweise simple Programmiersprache, die einfach gelernt werden kann, Was einigen Aufwand gespart hat. Allerdings ist Python keine allzu beliebte Programmiersprache in der Spieleentwicklung, da sie bei komplexeren Spielen schnell an die Grenzen ihrer Leistung stossen kann und es auch fast keine Unterstützung für dreidimensionale Spiele gibt.

Die Entscheidung, kein Game-Engine und stattdessen eine Python-Bibliothek zu benutzen, war sehr bewusst, da es im Kern des Informatikteils dieser Maturaarbeit auch darum geht, zu lernen wie Videospiele fundamental funktionieren.

4.2.2 Programmatische Umsetzung der kryptografischen Methoden

Für alle im Kapitel 3 vorgestellten Verschlüsselungsmethoden wurde je eine Klasse mit einer Verschlüsselungs- und einer Entschlüsselungsfunktion programmiert. Dafür wurden zuerst zwei allgemeine Funktionen `letterify()` und `numberify()` geschrieben, die Zahlenwerte in ihre entsprechenden Buchstaben, beziehungsweise Buchstaben in ihre entsprechenden Zahlenwerte zwischen 0 und 25 umwandeln können und es so ermöglichen mathematisch mit den Buchstaben umzugehen.

```
def numberify(letter):
   integer = ord(letter)

    if 65 <= integer <= 90:
        integer -= 65

    if 97 <= integer <= 122:
        integer -= 97

    return integer
```

```
def letterify(integer):
    integer += 97
    letter = chr(integer)

    return letter
```

Alle der zu den Verschlüsselungstechniken geschriebenen Klassen sind im Grunde gleich aufgebaut. Sie enthalten zwei Funktionen, `encrypt()`, die Verschlüsselung und `decrypt()`, die Entschlüsselung. Dabei werden den Funktionen als Parameter jeweils der Klar- beziehungsweise Geheimtext und falls die Verschlüsselungsmethode einen Schlüssel benutzt auch der Schlüssel übergeben. Danach wird eine Schleife so oft ausgeführt, bis sie alle Buchstaben des Klartextes durchgearbeitet hat.

```
class Caesar():
    def encrypt(self, plaintext, key):
        ciphertext = ""
        for char in plaintext:
            ciphertext + = letterify((numberify(char) + key) % 26)

        return ciphertext

    def decrypt(self, ciphertext, key):
        plaintext = ""
        for char in ciphertext:
            plaintext + = letterify((numberify(char) - key) % 26)

        return plaintext
```

Im Beispiel dieser Cäsar Verschlüsselung wird am Anfang die Variable `ciphertext`, ein String in den Schritt für Schritt der Geheimtext hineingeschrieben wird, initialisiert. Danach wird mit einer `for`-Schleife jeder Buchstabe im Klartext einzeln verschlüsselt und an den Geheimtext angefügt. Für den Code der Verschlüsselung wird die mathematische Darstellung der Cäsar-Verschlüsselung benutzt. Zuerst wird der Buchstabe mit `numberify()` in eine Zahl zwischen 0 und 25 umgewandelt, zu dem dann der Schlüsselwert `key` addiert wird. Das Resultat der Rechnung wird mit einer modulo-26 Rechnung (in Python dargestellt mit einem Prozentzeichen) wieder in eine Zahl zwischen 0-25 umgewandelt, für den Fall, dass das Resultat durch die Addition grösser als 25 wurde. Die daraus resultierenden Zahl wird mit letterify() wieder in einen Buchstaben umgewandelt und hinten an den Geheimtext angefügt.

```
class VariantBeaufort():
    def encrypt(self, plaintext, key):
        ciphertext = ""
        for i, char in enumerate(plaintext):
            ciphertext += letterify((numberify(char)
                                - numberify(key[i % len(key)])) % 26)

        return ciphertext

    def decrypt(self, ciphertext, key):
        plaintext = ""
        for i, char in enumerate(ciphertext):
            plaintext += letterify((numberify(char)
                                + numberify(key[i % len(key)])) % 26)

        return plaintext
```

Eine etwas kompliziertere Verschlüsselung zum Programmieren war beispielsweise die Beaufort Variante, da bei dieser nicht einfach ein konstanter Schlüssel addiert werden konnte, sondern aus dem Schlüssel erst einmal mit dem Index des Klartextbuchstabens der richtige Schlüsselbuchstabe herausgenommen werden musste. Da es aber Fälle gibt, in denen der Schlüssel kürzer ist als der Klartext, muss auch auf den Index eine Modulo-Rechnung angewendet werden, aber diesmal eine mit dem Wert der Anzahl Stellen, die der Schlüssel hat. So wird der Schlüssel so oft wiederholt, bis er die Länge des Klartextes abdeckt.

4.2.3 Einfache Erweiterung

Da die meisten Skripte dieses Projekts objektorientiert und in Form von Klassen programmiert wurden, ist es relativ einfach selbst neue Räume anzufügen. Sobald im Hauptmenü der Startknopf angeklickt wird, werden alle Raum- und Einrichtungsobjekte geladen. Dies wird mit der `reset_game()`-Funktion gemacht.

```
def reset_game():
    global uv_lamp, cipher_wheel, scene, rooms_list, room, color_theme
    import random, Tools, Room, Object, Cryptography

# die Zeit im Timer wird zurückgesetzt auf die Startzeit
    timer.time = default_timer_time
# Die UV-Taschenlampe und die Chiffrierscheibe werden neu geladen
    uv_lamp = Tools.UVLight()
    cipher_wheel = Tools.CipherWheel()
```

```
# Das Spiel wird gestartet und es wird zur Hauptansicht umgeschaltet
    scene = 1
# Die Zeit auf dem Timer wird gestartet
    timer.running = True

# alle Codes, Passwörter und Schlüssel für alle Räume werden geladen
    room1_code = str(random.randint(1000, 9999))
    room2_code = random.choice(words_list)
    room2_key = str(random.randint(1, 25))
    room3_code = random.choice(words_list)
    room3_key = random.choice(words_list)

# Alle Räume mit ihren Objekten werden generiert
    rooms_list = [
        Room.Room([
            Object.Shelf(0, 1),
            Object.Door(5, 0, room1_code),
            Object.Shelf(1),
            Object.Lamp(4),
            Object.Desk(2),
            Object.Desk(3, 1, room1_code)
        ]),
        Room.Room([
            Object.Door(0, 1, room2_code),
            Object.Shelf(5),
            Object.Lamp(1),
            Object.Desk(4, 2),
            Object.Calendar(2, room2_code, room2_key),
            Object.Desk(3)
        ]),
        Room.Room([
            Object.Desk(0, 4, room3_code),
            Object.Shelf(5),
            Object.Desk(1, 3),
            Object.Shelf(4, 2, room3_key),
            Object.Door(2, 0, '55555'),
            Object.Lamp(3)
        ])
    ]

# Der Raum wird zum ersten Raum in der Raumliste gesetzt
    room = rooms_list[0]
# Das Farbschema wird so gesetzt, dass die Lampe im Raum eingeschalten ist
    color_theme = light_theme
```

Um einen neuen Raum einzuprogrammieren muss also im Grunde genommen nur beim Laden aller Codes, Passwörter und Schlüssel der Code zum Entkommen des neuen Raumes eingefügt werden und es muss der neue Raum mit seinen Objekten in die `rooms_list` eingefügt werden und eventuelle neue Einrichtungsgegenstände müssen in der `Objects.py` Datei definiert werden.

5 Reflexion und Ausblick

5.1 Rückblick

Grundsätzlich bin ich mit meiner Arbeit zufrieden, allerdings gibt es auch noch viel Luft nach oben.

Mein Hauptproblem war das Zeitmanagement. Es fiel mir besonders am Anfang der Arbeit sehr schwer einzuschätzen, welcher Teil der Arbeit wie viel Zeit in Anspruch nimmt. So hatte ich zum Beispiel im Konzept geplant, die Programme und den schriftlichen Teil bis Ende der Sommerferien fertig geschrieben zu haben.

Auch die feinere Aufteilung der Arbeit bereitete mir beim Programmieren grosse Probleme. Anfangs hatte ich eine grosse Programmdatei mit etwas über tausend Zeilen, in der es schwierig war, die Übersicht zu behalten, besonders dann, wenn ich an mehreren Stellen gleichzeitig arbeiten musste. Das Ganze dann nach allen Regeln der Kunst in einzelne Skripte aufzuteilen, die trotzdem noch korrekt miteinander funktionieren, kostete mich ziemlich viel Zeit. Auch war ich ziemlich überfordert damit, alles bereits Programmierte noch einmal durchgehen und überprüfen zu müssen und gleichzeitig alles in Klassen umzuprogrammieren, die ich von den Funktionen und Syntax her noch nicht lange und auch noch nicht gut kannte. Das objektorientierte Programmieren (das Arbeiten mit Klassen und Objekten) war insgesamt noch ganz neu für mich und ich musste es zuerst einmal lernen.

Auch konnte ich meine ursprünglichen Vorstellungen nicht ganz verwirklichen. So war am Anfang die Idee, sich im Raum auch umdrehen oder sogar umhergehen zu können. Darauf musste ich leider verzichten, weil es programmatisch zu schwierig umzusetzen war.

Zuletzt war es eine zusätzliche Herausforderung, meine Dateien mit GitHub zu verwalten, sodass ich, im Falle eines Fehlers oder dem Verlust einer Datei auf alle hochgeladenen Versionen der Datei zugreifen und sie herunterladen könnte. Davon profitierte ich tatsächlich mehrmals, zum Beispiel als ich beim Aufteilen der Hauptdatei in einzelne Skripte meine Hauptdatei mit einem anderen Skript ersetzte und es so speicherte.

Rückblickend bin ich immer noch überzeugt von meiner Themenwahl, da mich die Kryptografie und das Spiele-Programmieren bis heute interessieren und ich deshalb immer mit Interesse arbeiten konnte.

5.2 Ausblick

Es gibt viele Dinge, die ich in meinem Spiel noch ausbauen oder besser machen könnte. Einerseits wäre das zum Beispiel schönere Grafik, etwas mehr als nur die schwarz-auf-weissen oder weiss-auf-schwarzen Linien. Bessere Grafik hätte mich zu viel Zeit gekostet und ich hätte dann wahrscheinlich eine ganze Maturaarbeit allein über die Grafik machen können, was nicht mein Ziel war. Ausserdem könnte man an einigen Orten auch Animationen verwenden.

Was ebenfalls noch fehlt für ein «vollständiges» Game, wären Musik und sonstige Geräusche, die ich nicht mit einbaute, da ich einerseits auch dafür nicht die Zeit gehabt hätte und andererseits fand ich sie für ein Game wie dieses auch nicht so wichtig.

Eine gute Möglichkeit für aufbauende und weiterführende Projekte steht durch die einfache Erweiterbarkeit des Spiels zur Verfügung. So könnte man mit wenig Aufwand zusätzliche Räume hinzufügen oder bereits Bestehende modifizieren.

Zuletzt könnte man beispielsweise auch die originale Idee, dass man sich im Raum frei bewegen kann, umsetzen oder Kontrolle mit der Tastatur ermöglichen, besonders bei den Türen, and denen man Wörter eingeben muss.

Abbildungsverzeichnis

Abbildungen ohne Quellenangaben wurden von der Autorin selbst erstellt.

Tabellenverzeichnis

Sämtliche Tabellen wurden vom Autor selbst erstellt.

Redlichkeitserklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt habe,
- dass ich auf eine eventuelle Mithilfe Dritter in der Arbeit ausdrücklich hinweise,
- dass ich vorgängig die Schulleitung und die betreuende Lehrperson informiere, wenn ich
  - diese Maturarbeit bzw. Teile oder Zusammenfassungen davon veröffentlichen werde

  oder
  - Kopien dieser Arbeit zur weiteren Verbreitung an Dritte aushändigen werde.
- dass mir das Merkblatt «Plagiat» sowie auch die Konsequenzen eines Plagiats bekannt sind.

Meine Maturaarbeit umfasst (ohne Titelblatt, Inhaltsverzeichnis, Redlichkeitserklärung und Abgabeinformationen, Quellen- und sonstigen Verzeichnissen und Anhang) xyz Zeichen (ohne Leerzeichen).

Ich gebe zu den Maturaarbeitsexemplaren folgende Gegenstände oder Produkte ab: Ein Datenträger mit den folgenden Inhalten:

- Den gesamten Quellcode des Spiels inkl. Grafiken und andere Dateien die dazugehören
- Eine kompilierte Version des Spiels (.exe Datei)
- Eine ReadMe-Datei, die erklärt welche Dateien wo dazugehören und wie sie benutzt werden.

Kriens, 29. September 2022

Medea Emch