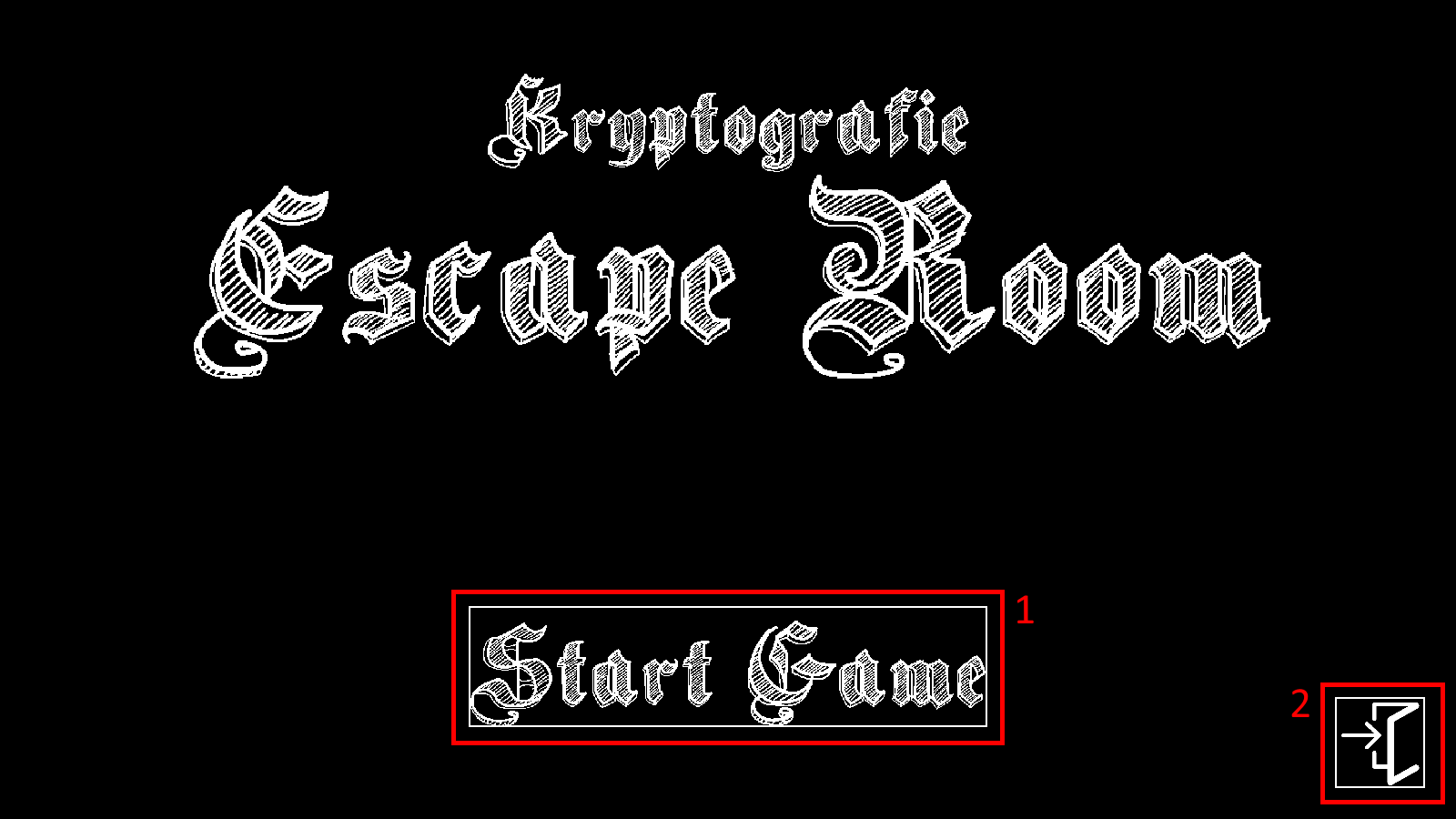
# Das Escape Room Spiel

Achtung: Dieses Kapitel enthält Informationen, wie das Spiel bedient und auch durchgespielt werden kann. Wer die Lösungen selbst herausfinden möchten, sollte dieses Kapitel vorerst überspringen.

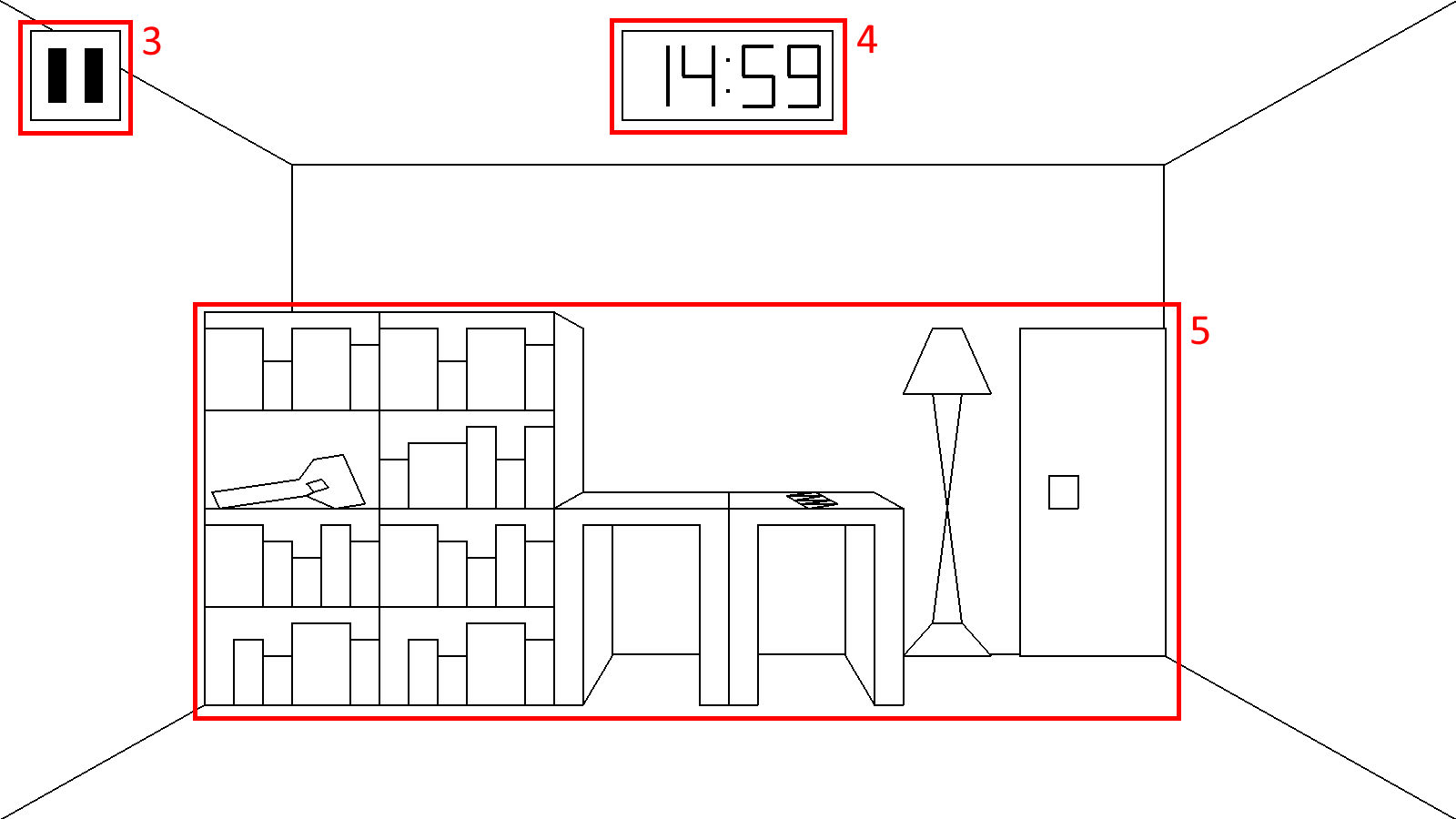
## Benutzeroberfläche

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

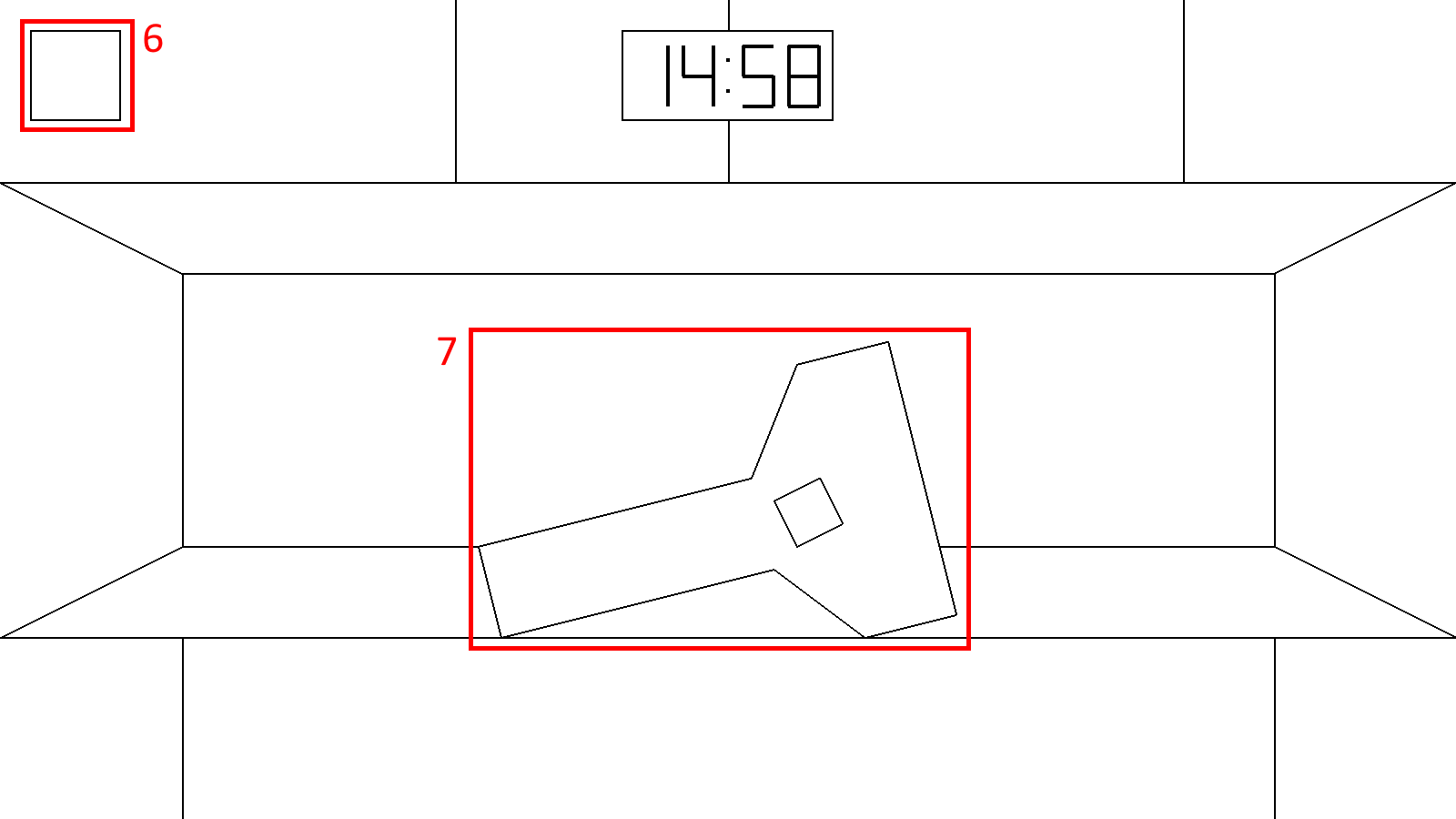


Im Hauptmenü gibt es zwei Knöpfe, den Startknopf (1), der das Spiel startet und den Spieler in den ersten Raum setzt und den Beenden-Knopf (2), der das Spiel stoppt und das Programm schliesst.

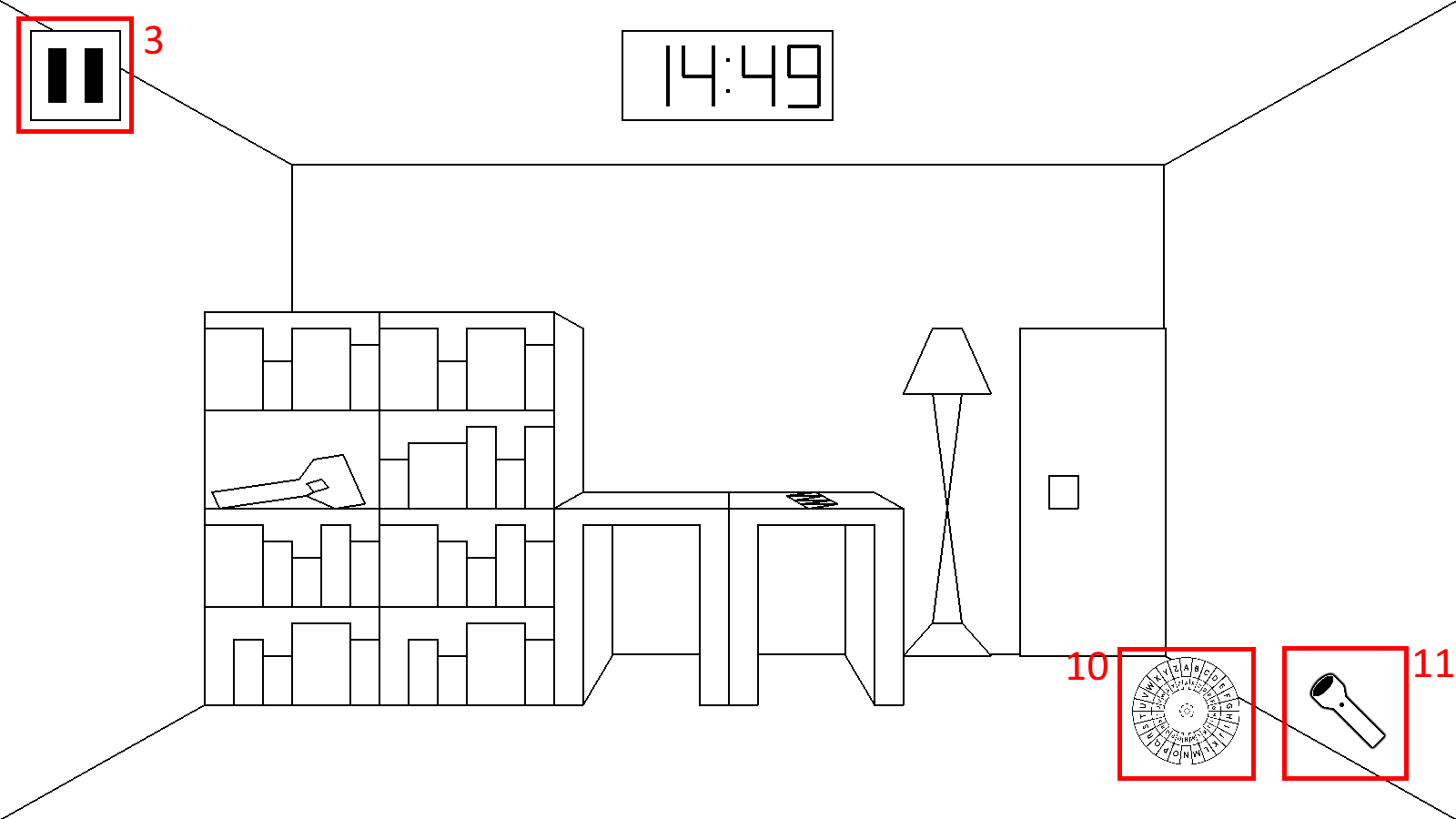
Das Spiel wird ausschliesslich mit der Maus gespielt, Tasteneingaben funktionieren nicht.



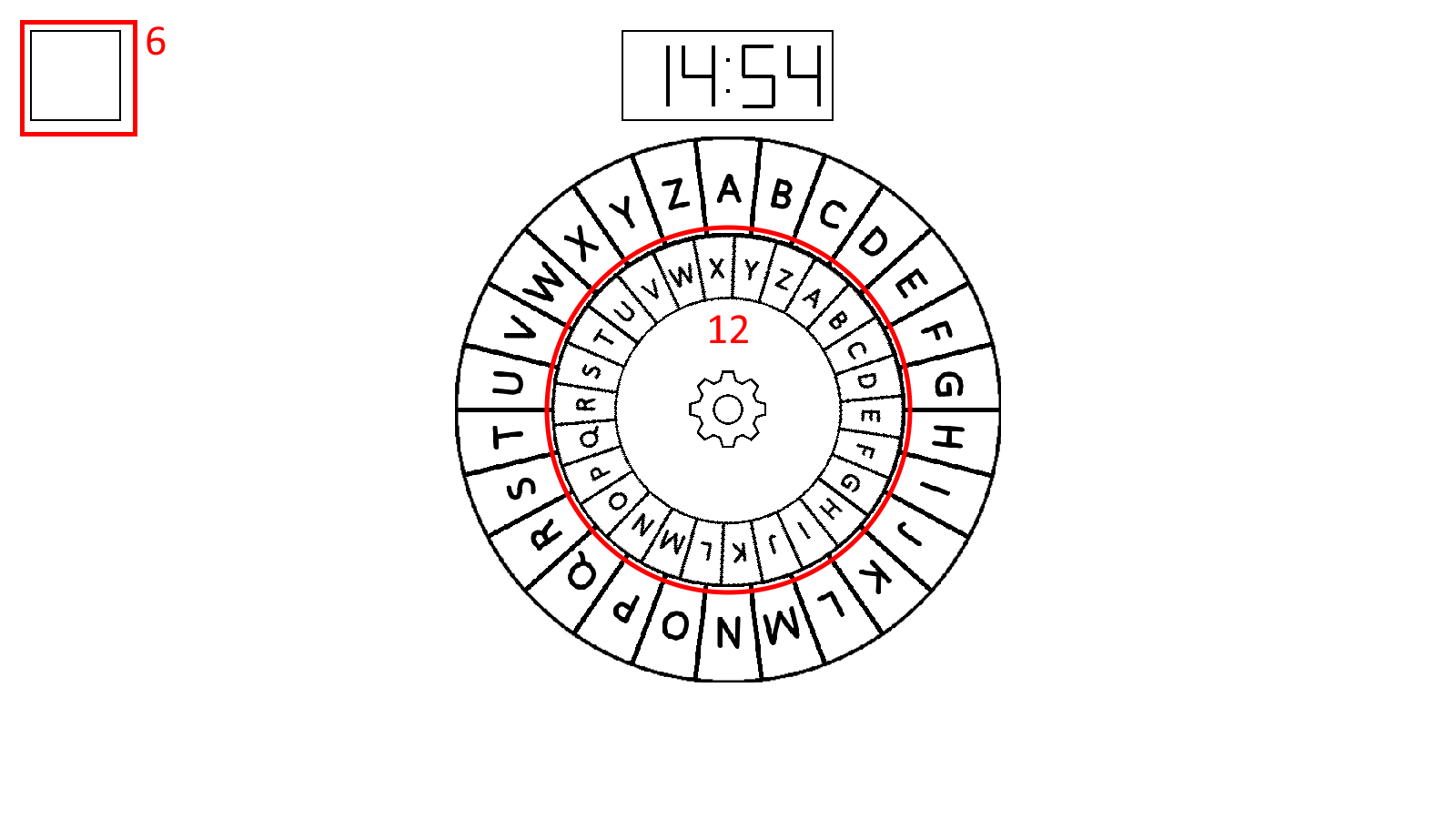
Wenn das Spiel gestartet wird, erscheint die Einrichtung des ersten Raumes (5), der Pausenknopf (3) und ein Timer (4). Das Ziel ist es, in allen Räumen das korrekte Passwort oder den korrekten Code einzugeben, bevor der Timer abläuft. Mit den meisten Objekten aus der Hauptansicht (5) kann interagiert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



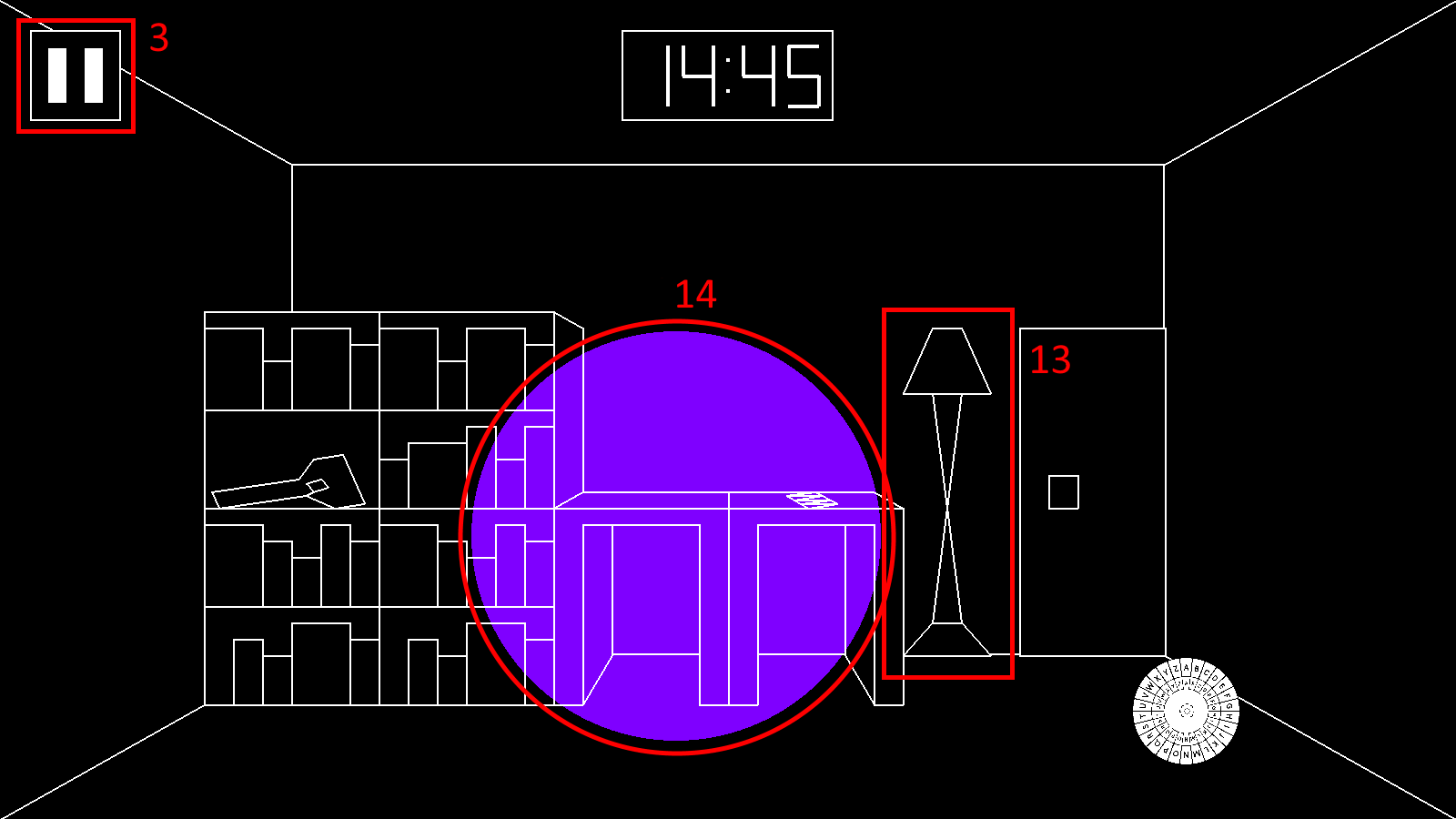
In den genaueren Ansichten von Objekten, wird der Pausenknopf durch einen Knopf (6) ersetzt, mit dem man zur Hauptansicht zurückwechseln kann. In den meisten Detailansichten von Objekten besteht die Möglichkeit weiter damit zu interagieren, so wie man beispielsweise hier die UV-Taschenlampe anklicken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



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

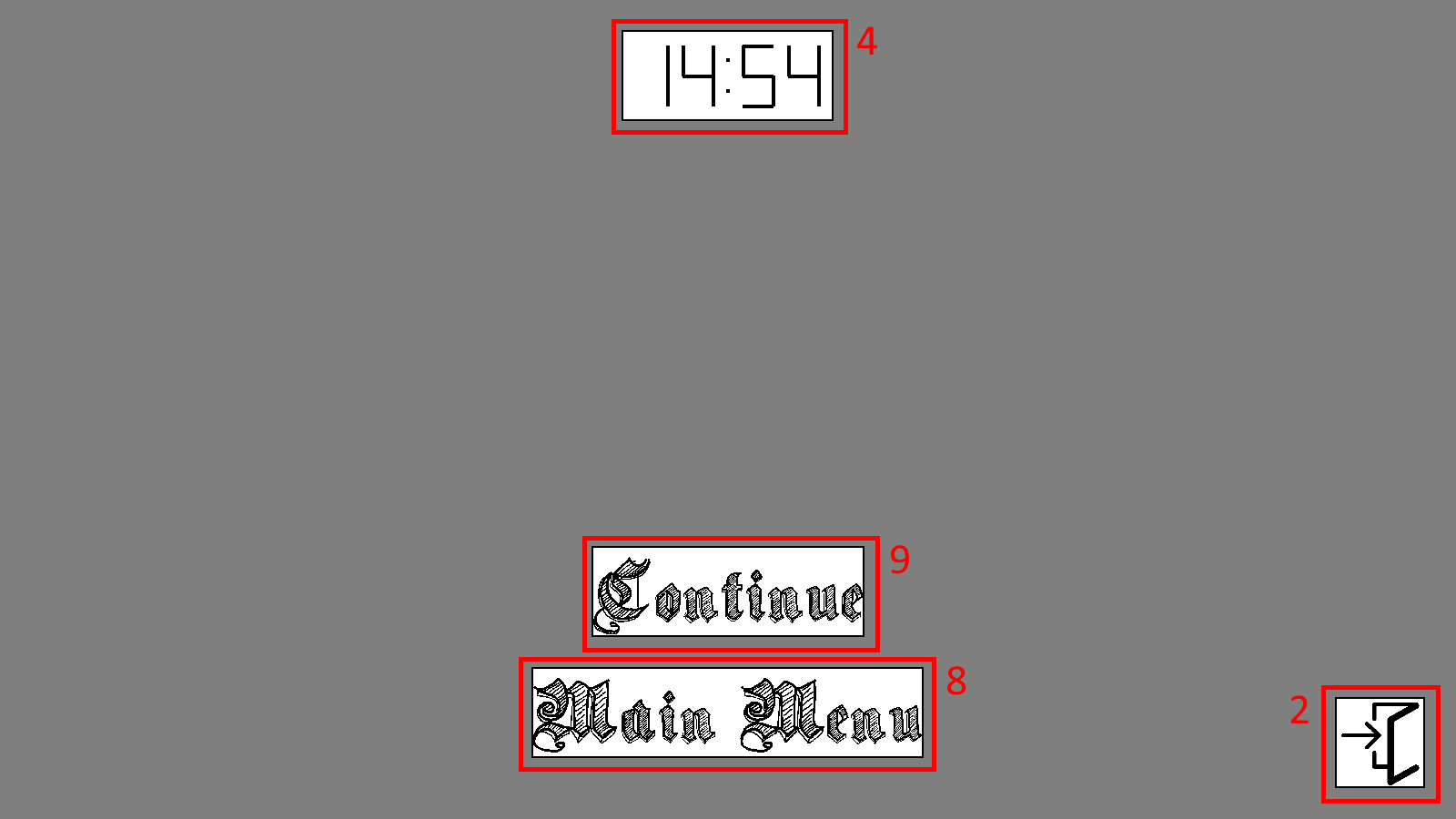


Im Falle der Chiffrierscheibe öffnet sich eine grosse Ansicht der Chiffrierscheibe, die auch benutzt werden kann. Dazu klickt man mit der Maus auf die innere Scheibe (12) und hält sie mit der Maus fest, während man daran dreht. Auch in dieser Ansicht befindet sich ein Knopf (6) mit der zurück zur Hauptansicht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



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.



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.

## Das Programm

### Wieso 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.

### 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.

### 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

timer.time = default\_timer\_time # die Zeit im Timer wird zurückgesetzt auf die Startzeit

uv\_lamp = Tools.UVLight()

cipher\_wheel = Tools.CipherWheel() # Die UV-Taschenlampe und die Chiffrierscheibe werden neu geladen

scene = 1 # Das Spiel wird gestartet und es wird zur Hauptansicht umgeschaltet

timer.running = True # Die Zeit auf dem Timer wird gestartet

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 Codes, Passwörter und Schlüssel für alle Räume werden geladen

rooms\_list = [ # Alle Räume mit ihren Objekten werden generiert

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)

])

]

room = rooms\_list[0] # Der Raum in dem sich der Spieler befindet wird zum ersten Raum in der Raumliste gesetzt

color\_theme = light\_theme # Das Farbschema wird so gesetzt, dass die Lampe im Raum am Anfang eingeschalten ist.

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.