# Projektdokumentation

Projekt

Infinity Spiegel

Ein Bild, das Elektronik, Computer, Leuchtdiode, Licht enthält.

Automatisch generierte Beschreibung

[Zusammenfassung 2](#_Toc171360773)

[Infinity-Spiegel 2](#_Toc171360774)

[Themenwahl 2](#_Toc171360775)

[Problemstellung 3](#_Toc171360776)

[Lösungsweg 3](#_Toc171360777)

[Erste Schritte 3](#_Toc171360778)

[Materialliste 3](#_Toc171360779)

[Bau des Spiegels 4](#_Toc171360780)

[Holzrahmen 4](#_Toc171360781)

[Halbdurchlässiger Spiegel 4](#_Toc171360782)

[Montage 4](#_Toc171360783)

[Implementierung 5](#_Toc171360784)

[Display 5](#_Toc171360785)

[LED-Lichtstreifen 5](#_Toc171360786)

[Display 6](#_Toc171360787)

[Einstellungen 6](#_Toc171360788)

[Standort 7](#_Toc171360789)

[Wetterdaten 9](#_Toc171360790)

[Displayanzeige 13](#_Toc171360791)

[LED-Lichtstreifen 17](#_Toc171360793)

[Initialisierung 17](#_Toc171360794)

[Farben 20](#_Toc171360795)

[Animationen 20](#_Toc171360796)

[Steuerung 24](#_Toc171360797)

[Quellen 25](#_Toc171360798)

# Zusammenfassung

## Infinity-Spiegel

Ein Infinity-Spiegel ist ein Spiegel, der die Illusion erzeugt in einen unendlich tiefen Raum zu schauen. Dieser Effekt wirkt sehr beeindruckend, sodass wir uns endschieden haben, ihn für unser Projekt zu nutzen. Der Spiegel wird mit steuerbaren LED-Lichtstreifen und einem Display ausgestattet, sodass wir unsere Software an diesen Geräten anwenden können. Das Display soll die Funktion besitzen, aktuelle Wetterdaten sowie den aktuellen Standort anzuzeigen. Die Wetterdaten enthalten ein Icon, den allgemeinen Wetterstatus, aktuelle Temperatur, Windrichtung und auch die Windgeschwindigkeit. Diese Daten und der aktuelle Standort sollen sich in einem definierten Zeitraum aktualisieren. Der LED-Lichtstreifen ist für den Infinity-Spiegel essenziel, da er zur Erzeugung der „Unendlichkeit Illusion“ beisteuert. Für diesen Effekt muss der Lichtstreifen das Licht zwischen den beiden Spiegeln erzeugen. Da das Licht den Effekt beeinflusst, sind sehr anschauliche „Spielereien“ mit dem Ansteuern des Lichtstreifens möglich.

# Themenwahl

Das Thema wurde aufgrund der zwei unterschiedlichen Anforderungen gewählt. Der erste Aspekt betrifft die Hardwareansteuerung des LED-Lichtstreifens und der Displayausgabe. Der Zweite Aspekt, ist die softwareabhängige Abfrage der Wetterdaten. Dies schien ein perfektes Projekt für die objektorientierte Programmierung zu sein.

# Problemstellung

Die Problemstellung dieses Projekts besteht darin, ein Display für den Infinity-Spiegel so zu programmieren, dass dieser periodisch Wetterdaten aktualisiert und sie auf eine ansprechende und leicht lesbare Weise darstellt. Des Weiteren muss der LED-Lichtstreifen so abgestimmt sein das der Infinity-Effekt hervorragend zur Geltung kommt. Da es mit einem Belichtungs-Modus zu monoton wird, werden hier mehrere Modi angestrebt. Zudem gibt es eine weitere, eigentlich nicht für das OOP-Projekt vorgesehene, weiter Aufgabe: Den Infinity-Spiegel zu bauen.

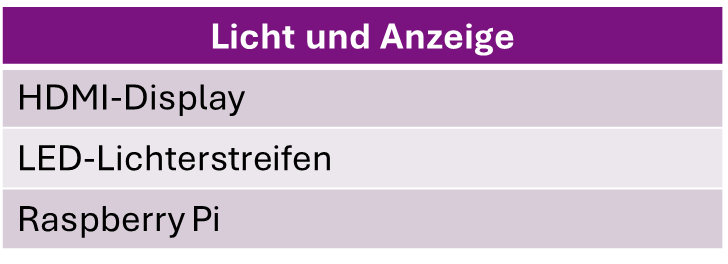
# Lösungsweg

## Erste Schritte

Als Erstes stellte sich die Frage, wo man einen Infinity-Spiegel herbekommt. Da wir schnell bemerkten, dass diese sehr teuer sind, entschlossen wir uns den Spiegel selbst zu bauen. Somit fuhren wir in den Baumarkt kauften alle benötigten Teile und bauten unseren eigenen Infinity-Spiegel.

## Materialliste





## Bau des Spiegels

### Holzrahmen

Zunächst sägten wir das Holz auf die Länge und Breite des gekauften Spiegels zurecht. Anschließend musste das Holz noch geschliffen werden, um eine glatte und gerade Oberfläche zu erzielen. Als dass gemacht war konnten wir die Holzbretter zu einem Rahmen verschrauben.

### Ein Bild, das Gelände, Metall, Werkzeug, Hartwaren enthält. Automatisch generierte BeschreibungHalbdurchlässiger Spiegel

Um diesen selbst zu bauen, benötigten wir nur ein wenig Plexiglas und eine Spiegelfolie, die auf einer Seite sichtdurchlässig ist und auf der anderen nicht. Als erstes sägten wir das Plexiglas auf die Größe des gekauften Spiegels. Daraufhin musste das Plexiglas nur noch mit der Folie foliert werden. Somit war der halbdurchlässige Spiegel schon fertig.

Ein Bild, das Rechteck, hölzern, Kunst, Fenster enthält.

Automatisch generierte Beschreibung

### Montage

Nun mussten wir nur noch den Rahmen, den halbdurchlässigen Spiegel und den normalen Spiegel zusammenbauen. Dafür nutzten wir Montagekleber und Klebeband. Der Rahmen dient gleichzeitig auch als Abstandshalter der beiden Spiegel, sodass das Licht genug Platz hat, um sich zu reflektieren.

## Implementierung

### Display

Die Implementierung des Displays war sehr einfach da wir ein HDMI-Display verwendeten. Das bedeutete wir mussten einfach den Raspberry Pi und das Display via HDMI-Kabel verbinden.

### Ein Bild, das Kabel, Elektronik, Elektrische Leitungen, Schaltung enthält. Automatisch generierte BeschreibungLED-Lichtstreifen

Um den LED-Streifen zu betreiben, löteten wir zuerst ein Netzteil an, sodass dieser nicht über den Raspberry Pi versorgt wird, da der Raspberry Pi nur zur Ansteuerung dient. Um dies zu ermöglichen, verkabelten wir die GPIO-Pins des Raspberry Pi mit dem LED-Streifen. Zusätzlich schlossen wir einen Knopf an, um den Modus wechseln zu können.

## Display

Für das Display haben wir eine Bibliothek entwickelt, die verschiedene Klassen umfasst. In diesen Klassen werden Einstellungen gespeichert und erstellt, der Standort in Koordinaten umgewandelt und Wetterdaten über eine API angefordert.

### Einstellungen

Für die wichtigsten Einstellungen, die auch gespeichert werden sollen, haben wir die Klasse Settings erstellt.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Bei der Initialisierung der Klasse über die init-Funktion wird das Attribut „user“ initialisiert. Anschließend wird überprüft, ob bereits eine Datei mit dem Benutzernamen und den dazugehörigen Einstellungsdaten existiert. Falls eine solche Datei nicht vorhanden ist, wird die set\_location\_settings-Funktion aufgerufen, die automatisch eine neue Datei mit dem Benutzernamen erstellt. In diesem Fall wird der Benutzer aufgefordert, die benötigten Daten über ein Eingabefeld anzugeben.

Für das Anlegen der neuen Daten haben wir die Funktion „set\_location\_settings“ erstellt. Wie oben beschreiben, frägt diese die Daten ab und speichert es in eine Json-Datei mit dem Namen des aktuellen Users.

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

Automatisch generierte Beschreibung

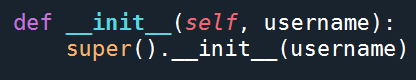
Fazit Settings-Klasse:

Die Klasse ist in der Lage, Einstellungen zu speichern und dient somit als ein wichtiges Tool für viele Programme. Sie kann zudem erweitert werden, um weitere Daten zu speichern.

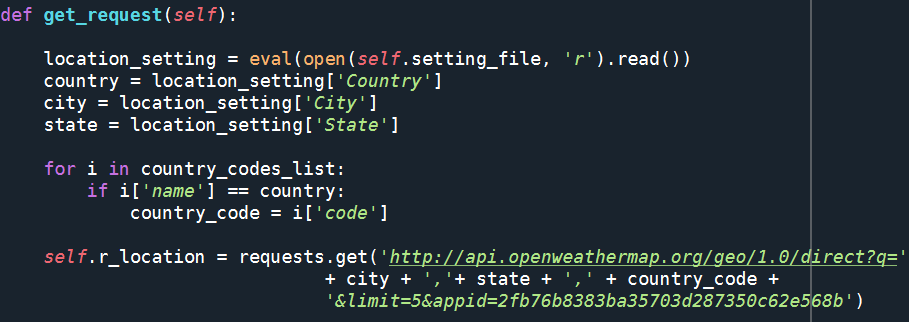
### Standort

Für die Wetterdaten wird eine kostenlose API verwendet. Diese gibt die Daten der nächstgelegenen Wetterstation, die dem Server zur Verfügung steht, aus. Um die Wetterdaten zu erlangen, muss man eine get-request an die API schicken. Für diese get-request benötigt man jedoch noch die Breiten- und Längengrade, um den Standort festzulegen. Somit haben wir die Klasse „Location“ erstellt, um anhand den Einstellungsdaten genaue Koordinaten zu erhalten.

Diese Klasse erbt von der Settings-Klasse, da diese immer initialisiert werden muss, wenn die Location-Klasse verwendet wird. Damit dies geschieht, wird der Befehl „super().\_\_init\_\_()“ in der init-Funktion von der Location-Klasse verwendet. Da das Attribut „user“ auch vererbt wird muss man diesen auch mit angeben.



Um an die Koordinaten zu kommen haben wir die Funktion „get\_request“ geschrieben.



Diese verwendet auch eine API. Als erstes muss eine Request gesendet werden. Um dies zu tun, benötigt man die Daten aus der Setting-Datei. Nach aufrufen dieser Datei werden die Informationen in den Attributen gespeichert. Die Attribute werden erst hier initialisiert, da sie bei jeder neuen Anfrage aus der aktuellen Settings-Datei abgefragt werden. Würden die Attribute in der init-Funktion initialisiert, blieben sie bei Änderungen in der Settings-Datei unverändert, da die Initialisierung nur einmal beim Erstellen der Instanz erfolgen würde. Diese Vorgehensweise stellt sicher, dass Änderungen in der Settings-Datei sofort wirksam werden und die Attribute bei jeder neuen Anfrage aktualisiert werden. Dann wird noch das benötigte Kürzel für das eingegebene Land anhand der for-Schleife gesucht und auch in ein Attribut gespeichert. Nun kann die Request gesendet werden. Von der Request bekommen wir eine Json-Datei als Antwort. Diese kann ähnlich wie ein Dictionary behandelt werden.

Um die Koordinaten aus der Request zu bekommen haben wir die Funktion „get\_coordinates“ geschrieben.

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

Diese nutz die get\_request-Funktion und sucht anhand dem Key „lat“ und „lon“ die passende Values. Diese stehen für die Breiten- und Längengrade.

Eine weiter Funktion namens „get\_city“ gibt die nächste Stadt der Koordinaten aus.

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

Fazit Location-Klasse:

Diese Klasse kann für alle möglichen Anwendungen von gebrauch sein, in denen genaue Koordinaten für einen festen Ort benötigt werden. Somit ist diese Klasse sehr universell.

### Wetterdaten

Für die Wetterabfrage wurde die Klasse Weather erstellt. Bei der Initialisierung wird eine Instanz der Location-Klasse übergeben, um deren Funktionen nutzen zu können. Anschließend wird getestet, ob wir eine Request an den Server mit den Wetterdaten stellen können.

Ein Bild, das Text, Screenshot, Schrift enthält.

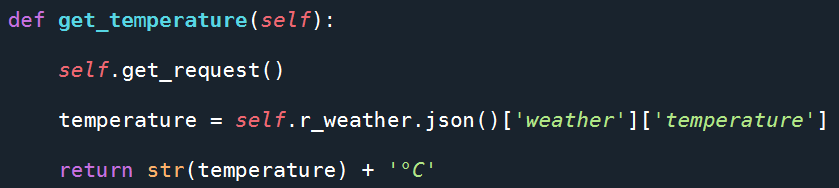
Automatisch generierte Beschreibung

Um eine Request an die API zu senden haben wir die Funktion „get\_request“ erstellt.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Die Funktion frägt die Koordinaten über get\_coordinates-Funktion ab. Diese werden dann in einen String konvertiert, um sie in die request.get-Funktion einfügen zu können. Die request.get-Funktion gibt eine Art Dictionary mit dem Datentyp Json aus. Diese Json Datei wird dann in einem Attribut gespeichert.

Für die Temperatur haben wir die Funktion get\_temperature erstellt.

Diese Funktion führt die get\_request-Funktion aus. Wandelt dann die Json-Datei in ein Dictionary um und sucht nach dem Value für den Key „temperature“. Dieser wird dann mit return zurückgegeben, davor wird er aber noch in einen String konvertiert und ein „°C“ hinzugefügt.

Für die Winddaten haben wird die Funktion get\_wind geschrieben.

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

Automatisch generierte Beschreibung

Diese Funktion führt auch die get\_request-Funktion aus. Wandelt diese in ein Dictionary um und sucht nach dem Value vom Key „wind\_speed\_30“ und den Key „wind\_gust\_direction\_30“. Daraus erhalten wir die Windgeschwindigkeit in km/h der letzten 30 Minuten und die Windrichtung in Grad in den letzten 30 Minuten. Damit die Windrichtung nicht in Grad angezeigt wird, sondern die ungefähre Himmelsrichtung, durchläuft die Windrichtungsinformation in Grad noch die If-Schleifen. Diese werten aus in welchem Intervall die Windrichtung liegt und speichert die passende Himmelsrichtung in der Variable wind\_direction ab. Diese wird mit return zurückgegeben zusammen mit dem String „Wind“. Desweiterem wird die Windgeschwindigkeit auch mit zurückgegeben.

Für den Status wird die Funktion get\_status verwendet.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Diese Funktion führt die get\_request-Funktion aus und sucht nach dem Value vom Key „Icon“. Dieses Value wird dann zurückgegeben.

Für das passende Icon würde die Funktion get\_icon erstellt.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Das Funktionsprinzip ist dasselbe wie bei der get\_status-Funktion. Nur muss diese noch das Value in den Path einsetzen, so dass dieser übergeben werden kann. Mit öffnen der Datei über den Path wird das passende Icon mit dem Datentyp JPG aufgerufen.

Die Icons wurden in dem weather\_icons-Ordner abgespeichert. Zudem wurden noch passende Dateinamen gewählt, die mit dem Status übereinstimmen, sodass es die Programmierung erleichterte.

Ein Bild, das Screenshot, Text, Design enthält.

Automatisch generierte Beschreibung

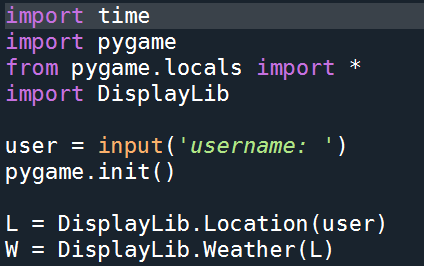
<https://www.iconfinder.com/search?q=weather>

Fazit Weather-Klasse:

Die Klasse Weather kann somit für alle Anwendungen die Wetterdaten benötigt verwendet werden. Außerdem kann sie noch erweitert werden für weitere Wetterdaten, wie zum Beispiel den aktuellen Luftdruck.

### Displayanzeige

Für die Displaydarstellung wurde eine bereits vorhandene Bibliothek verwendet namens Pygame. Diese ermöglicht Grafiken anzuzeigen und zu animieren. Diese wird in das Hauptprogramm importiert so wie unsere Bibliothek. Dann muss die Klasse pygame.init() und unser Weather- und Location-Klasse einmal initialisiert werden. Davor wird noch der Username abgefragt, um dies an die Location-Klasse zu übergeben.



Danach werden die Skalierungen für das Icon festgelegt sowie den Skalierungsfaktor für die Animation. Als nächstes fragen wir den passenden Icon-Path zu den aktuellen Wetterdaten, anhand unserer get\_icon-Funktion ab. Nun werden Text-Objekte eingefügt, den passenden Text laden wir über unsere Funktionen aus der Weather-Klasse. Als letztes legt man noch die Größe des Programmfensters fest, hier verwenden wir den Vollbildschirm.

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

Automatisch generierte Beschreibung

Im Hauptprogramm gibt es eine Hauptschleife und eine Nebenschleife. Das Programm läuft eine festgelegte Zeit in der Hauptschleife, wenn diese unterbrochen wird, werden alle Daten anhand der Funktionen in der Weater-Klasse aktualisiert.

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

Automatisch generierte BeschreibungEin Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Ausgabe:

## 

## LED-Lichtstreifen

Für den LED-Streifen haben wir ebenfalls eine Bibliothek entwickelt, die verschiedene Klassen umfasst. In diesen Klassen werden Informationen über den LED-Streifen gespeichert und verarbeitet, Farben erzeugt und die Ansteuerung über einen Druckknopf gewährleistet.

### Initialisierung

Für die Initialisierung des LED-Streifens wird die Klasse PixelStrip im Programm „*rpi\_ws281x*“ verwendet. Dieses stammt aus der Bibliothek „*rpi-ws281x-python*“. Die

Klasse PixelStrip war bereits objektorientiert gestaltet und wurde von uns übernommen. Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Karte Menü, Schrift enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Zusammenfassend dient die Klasse *PixelStrip* als Treiber für WS281x LED-Streifen in Python. Sie ermöglicht die Initialisierung und Steuerung der LEDs über einen GPIO-Pin. Parameter wie die Anzahl der LEDs, die Frequenz des Anzeigesignals und die Helligkeit sind durch die Klasse konfigurierbar. Sie definiert Methoden zum Setzen und Lesen von RGB-Farbwerten der LEDs an bestimmten Positionen, zur Steuerung der Helligkeit und zur Aktualisierung des LED-Streifens.

### Farben

Die Klasse *PixelStrip* ruft bei der Steuerung der Farben die Klasse *Color* auf, die sich ebenfalls in dem Programm *rpi\_ws281x* befindet.

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

In dem ursprünglichen Programm war die Methode *RGBW* eine Klasse und die Klasse *Color* eine Methode. Dies haben wir getauscht, um das Programm übersichtlicher zu machen und damit beim Aufrufen klarer wird, welche Methode genau durchlaufen wird und wo diese zu finden ist.

Die Klasse Color dient also dazu, die gewünschte Farbe, die beim Aufrufen der Animationen im RGB-Format (0, 0, 0) angegeben wird, für die Ansteuerung entsprechend auszugeben.

### Animationen

Für die Animationen ist die Klasse *LEDAnimation* im Programm *led\_strip\_animations* zuständig. Dieses Programm stellt auch unser main-Programm dar, das aufgerufen wird, um den LED-Streifen anzusteuern. Hier werden also alle Klassen zusammengeführt und schließlich ausgeführt.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Die Klasse *LEDAnimation* erbt von den Klassen *PixelStrip* und *Color*. Die Parameter, die in der Klasse *PixelStrip* verarbeitet werden, werden bei der Initialisierung eines Objekts der Klasse *LEDAnimation* gespeichert.

Darüber hinaus werden in der Klasse *LEDAnimation,* wie der Name schon sagt, die Animationen, die auf dem LED-Streifen ablaufen sollen, gespeichert. Die Animationen haben wir auch von der Bibliothek auf Github übernommen.

Ein Bild, das Text, Screenshot, Karte Menü enthält.

Automatisch generierte Beschreibung

Hier wird auch nochmal die Verarbeitung der Farben über die Klasse *Color* mit der Methode *RGBW* deutlich.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Insgesamt gibt es 6 verschiedene Animationen.

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

Automatisch generierte Beschreibung

Im main-Programm wird mit den entsprechend erwarteten Parametern ein Objekt der Klasse *LEDAnimation* erzeugt und aufgerufen.

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung

Zum Durchschalten der verschiedenen Animationen wird eine globale Variable *animation* erzeugt und durch Zeile 98 zwischen den Werten 0 und 5 gehalten. Jeder Animation wird jetzt ein Wert zwischen 0 und 5 zugeordnet. Bei Betätigung des Druckknopfs soll sich der Wert der Variable *animation* um Eins erhöhen.

### Steuerung

Hier kommt dann die Klasse *Button* im Programm *button* zum Einsatz.

Ein Bild, das Text, Schrift, Screenshot enthält.

Automatisch generierte Beschreibung

Diese wird im Hauptprogramm *led\_strip\_animations* importiert.

Ein Bild, das Text, Software, Multimedia-Software, Grafiksoftware enthält.

Automatisch generierte Beschreibung

Das Programm dient dazu, Signale von einem Knopf zu verarbeiten. Das *RPi.GPIO*-Modul wird importiert, um die GPIO-Pins des Raspberry Pi anzusteuern. Die Klasse *Button* beinhaltet die benötigten Funktionen. Wenn die Klasse erstellt wird, wird der GPIO-Pin (pin), an dem der Knopf angeschlossen ist, festgelegt. Zudem gibt es eine Rückruffunktion (callback), die aufgerufen wird, wenn der Knopf gedrückt wird. Um Störungen zu vermeiden, wird noch eine Entprellzeit (bouncetime) definiert.

Der GPIO-Pin wird als Eingang festgelegt. Beim Drücken des Knopfs wird die interne Methode \_internal\_callback ausgelöst. Diese Methode ruft die Rückruffunktion auf. Die cleanup-Methode setzt den GPIO-Pin zurück, damit keine Ressourcen verschwendet werden, wenn der Knopf nicht mehr gebraucht wird.

# Quellen

Bauanleitung Infinity Spiegel:

<https://www.obi.de/magazin/mach-mal-mit-obi/infinity-mirror-selber-bauen>

Icons:

<https://www.iconfinder.com/>

Display Bibliothek Pygame:

<https://www.pygame.org/>

LED-Lichtstreifen:

<https://github.com/rpi-ws281x/rpi-ws281x-python>

Knopf:

<https://tutorials-raspberrypi.de/raspberry-pi-kamera-per-gpio-knopfdruck-ausloesen/>