SELF-INFLICTED EMPATHY

Wie lässt sich emotionales Erleben in der Mensch-Maschine-Kommunikation erfahrbar machen?

Konzeption und Umsetzung eines Application Programming Interface mit dem Schwerpunkt automatisiertes emotionales Verständnis von nutzer*innengenerierten Texten

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences

SELF-INFLICTED
EMPATHY

BACHELORARBEIT

Game Design
Fachbereich 5 -
Gestaltung und Kultur

vorgelegt von:
Caspar Schirdewahn

Berlin, 29.06.2020

Erstgutachter:
Prof. Thomas Bremer

Zweitgutachterin:
Prof. Susanne Brandhorst

SELF-INFLICTED EMPATHY

VORWORT

SELF-INFLICTED EMPATHY ist ein exploratives Bachelorprojekt, das sich mit dem emotionalen Erleben von maschinell interpretierten, nutzer*innengenerierten Textinhalten auseinandersetzt. Basierend auf zeitgemäßen Machine-Learning Algorithmen soll diese künstlerisch-technische Arbeit die kreativen Instrumente des “Emotional Understanding” ausloten und durch prototypische Inszenierung verdeutlichen.

Neben dem Ausblick auf das Mögliche, liegt dabei aber auch das Unmögliche, oder besser das Untragbare im Fokus. Technologische Innovationen, besonders im Bereich der Kommunikation, ziehen in der Regel gesellschaftspolitische Diskurse nach sich, die neben Verhaltensregeln und Gewohnheiten auch immer wieder Persönlichkeitsrechte verhandeln. Während sich die Frequenz richtungweisender technologischer Neuerungen stetig verdichtet, wird auf der anderen Seite die Wissens- und Verständniskluft zwischen Entwickler*innen und Nutzer*innen immer größer - die gesellschaftlichen, moralischen und sozioökonomischen Implikationen immer undurchsichtiger. Aus diesem Grund war es mir wichtig, neben dem spielerischen Gehalt auch potentielle Gefahren und systemische Folgeeffekte in meine Evaluation miteinzubeziehen.

Die algorithmisierte Interpretation von Emotionen steckt auf der technischen Ebene zwar nicht mehr in den Kinderschuhen, ist allerdings von den Marktführern in den meisten Fällen ein durch Closed Source stark behütetes Geheimnis. Zielgruppe für diese Serviceanbieter sind zumeist größere Firmen, die sich von der neuen Technologie entweder die Automatisierung von firmeninternen Feedbackloops, Marken-Assoziationsanalyse in sozialen Netzwerken oder die Identifikation problematischer Schritte in ihrer Customers-Journey versprechen.

Mit zunehmenden Open Source Alternativen und kostenfreien APIs wird in Zukunft auch die Zahl an Spielen und interaktiven Projekten steigen, die sich dieser Technologie bedienen werden, um einzigartige, immersive Erfahrungen zu erschaffen. Diese Bachelorarbeit möchte sich im Sinne einer Potenzialanalyse, der Exemplifikation von Nutzungsfällen und des Barriere-Abbaus für Developer*innen verstanden wissen. Das Designprojekt SELF-INFLICTED EMPATHY war eine anregende Untersuchung der technischen Möglichkeiten und der daraus resultierenden Design-Implikationen für das Entwicklungsfeld der fühlenden Maschine.

Neben der Dokumentation der Arbeitsergebnisse und des Workflows, technischen Hürden und daraus gewonnenen Erkenntnissen, soll die Evaluation und der Ausblick auch mögliche Anwendungsfälle für die durch diese Arbeit entstandene API behandeln. Der explorative, prototypische Charakter des Projekts hat über die Entwicklungszeit jede Menge Möglichkeiten für bedeutende Folgeprojekte aufgetan, die von medien-künstlerischen Ansätzen bis zu klassischen Video-Spiel Features reichen.

Möglich wurde der, über die Projektzeit gewachsene, Fokus der Arbeit und die reichhaltigen Ergebnisse auch durch das substanzielle Feedback meines betreuenden Professors. Eine umfangreiche Danksagung ist im Anschluss an die Projektdokumentation zu finden.

EXPOSÉ - KURZFASSUNG

Große Teile unserer Alltagskommunikation haben sich in den letzten 15 Jahren in digitale Strukturen eingegliedert. Virtuelle Avatare in Social Media Netzwerken sind für viele Menschen Kernstück ihrer emotionalen Expression und Selbstdarstellung. Der Transfer sozialer Strukturen, gesellschaftlicher Zwänge und Zugehörigkeitsmuster in die permanente Erreichbarkeit des Internets hat Persönlichkeitsmerkmale und Gemütszustände zu optimierungsbezogenen Statussymbolen verklärt.

Ich interessiere mich schon seit geraumer Zeit für die Semantik medial vermittelter Emotionen. Lineare Medien haben zahlreiche Methoden hervorgebracht Emotionen zu präsentieren und zu vermitteln. Die Reflexivität von Spielen und interaktiven Erfahrungen ermöglicht hingegen die Emotionen der Rezipient*in selbst in den Mittelpunkt des Erlebens zu stellen.

In meiner künstlerisch-technischen Arbeit möchte ich daher nutzer*innengenerierte emotionale Inhalte in die Intimität der Mensch-Maschine-Kommunikation überführen und zu einem Moment der Selbstwahrnehmung stilisieren. Für die technische Umsetzung möchte ich verschiedene open source APIs explorieren, die, durch die Interpretation von Eingaben, Wahrscheinlichkeitsprognosen ausgeben, welche für die Produktion eines modularen, emotionalen Kategoriensystems genutzt werden können.

Vor allem die emotionale Verortung und die fundierte, angemessene Zuweisung in passende Interpretationssysteme, sowie deren Übersetzung in interaktive Regelwerke bilden den Kern meiner Arbeit. Die kategorisierten Ergebnisse sollen anschließend in Spielelementen Verkörperung finden, welche durch bspw. Verhalten, Regelwerk oder Visualisierung die Gefühlswelt des Nutzenden widerspiegeln und in eine individuelle, ästhetische Erfahrung überführen.

Aktuell kann ich mir als Endgerät gut ein aktuelles Smartphone vorstellen, da es reihenweise Sensoriken bietet, deren Parameter über einen rein kognitiven Gehalt der Eingabe hinausgehen. Des Weiteren sind Handys in ihrer Gebrauchsweise bereits etablierte persönliche Alltagsbegleiter, die durch ihre taktile Bedienung einen besonderen Grad der Zuneigung und Produktbindung evozieren.

Die Applikation, die als Werkstück meiner Bachelorarbeit entstehen soll, könnte somit den Charakter eines lebhaften, emotionalen Tagebuchs bekommen. Ich sehe in dem Design einer solchen interaktiven Erfahrung eine besondere Herausforderung, da klassische, zielorientierte Gameloops voraussichtlich schlicht als Ideenstifter, nicht aber als Fundament der Handlungsmotivation implementiert werden können.

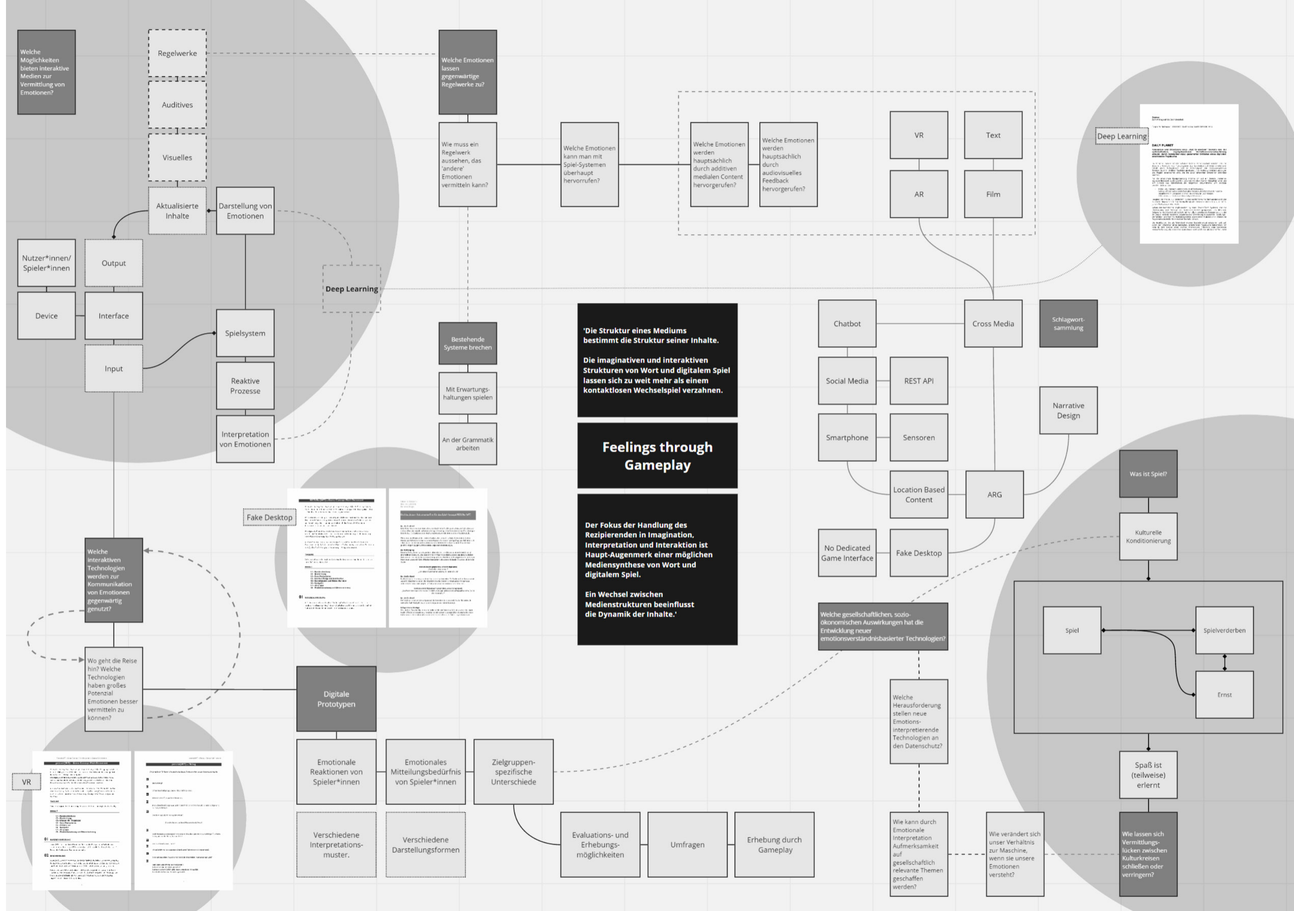


Abb. 2: Ideationboard, Miro-Board Screenshot

MOTIVATION

Mein Interesse an Medienstrukturelementen, deren Auswirkung auf Rezeption, Gestaltung und zwischenmenschliche Kommunikation begleitet mich bereits seit meinem ersten Bachelorstudium der Kommunikationswissenschaft und Philosophie. Experimentelle Auseinandersetzungen mit Radio-, Text- und Filmproduktionen waren in den Jahren vor meinem Game Design Studium an der HTW Berlin weitere Bereiche, in denen ich diesbezüglich Erfahrungen sammeln konnte.

Neben der Auswirkung menschengenerierter Inhalte auf andere Menschen habe ich im Game Design Studium viel Kontakt zu maschinellen Interpretationsebenen von nutzer*innengenerierten Inhalten gehabt. Letztlich sind digitale Spiele immer reaktive Systeme, die versuchen Spieler*inneninput bestmöglich zu deuten und einen auf Regeln basierten adäquaten Output zu generieren.

Alle Medienbereiche, mit denen ich mich bisher eingehend beschäftigt habe, besitzen ihre eigenen Kniffe und Methoden, um emotional bewegende Inhalte zu schaffen und zu transportieren. Interaktive, computergestützte Erfahrungen können viele unterschiedliche Medienformen inkorporieren und aus dem vorhandenen methodologischen Erfahrungsschatz schöpfen. Darüber hinaus ist es ihnen aber auch möglich, neue Formen der Vermittlung zu finden, die auf dem sich stetig erweiterndem Entwicklungshorizont technologischer Sensorik und algorithmisierter Interpretation basieren.

Mein beständiges Interesse an außergewöhnlichen Kommunikationssystemen hat mich zum Studium Game Design geführt und da meine Begeisterung für diesen Themenbereich nicht im Geringsten abgenommen hat, habe ich mich in meiner Bachelorarbeit mit dem kontemporären Feld der Mensch-Maschine Kommunikation auseinandergesetzt.

UMFANG DES WERKSTÜCKS

Der Inhalt des Designprojekts, welches in dieser Arbeit dokumentiert ist, stellt das Werkstück meiner Bachelorarbeit dar. Durch den explorativen Charakter meiner Arbeit sind zu den, ursprünglich im Exposé aufgeführten, Ergebnissen noch weitere hinzugekommen. Mit der Abgabe dieses Projektbuches sind enthalten:

- **Unity Projektordner des Prototypen**
- **Spielbare APK des Prototypen**
- **Video der APK in Benutzung**

- **Unity Projektordner der selbst entwickelten API**
- **Unity Package der selbst entwickelten API**
- **Video der selbst entwickelten API in Benutzung**
- **Schriftliche Dokumentation und Nutzungs-Anleitung der selbst entwickelten API**

- **20 chronologische Screenshots der Entwicklung des Prototypen und der API**
- **10 Making-of Abbildungen**
- **3 Klassendiagramme, zwei davon von Prototypen, eines von der API**

Die APK des letzten Prototypen vor der Entwicklung der API ermöglicht es Nutzer*innen durch die Eingabe kurzer Texte eine farblich atmende Kachelwand zu beeinflussen. Die in dem Text erkannten Emotionen wirken sich auf Farbigkeit und die Überblendung der Kacheln aus. Diese Umsetzung zeigt exemplarisch auf, was mit der im Anschluss entwickelten, überarbeiteten API möglich ist. Im Ausblick dieser Bachelorarbeit finden sich diesbezüglich mehrere Überlegungen und Anstöße, die die Nutzung von Emotional Understanding in verschiedenen Entwicklungskontexten beleuchten.

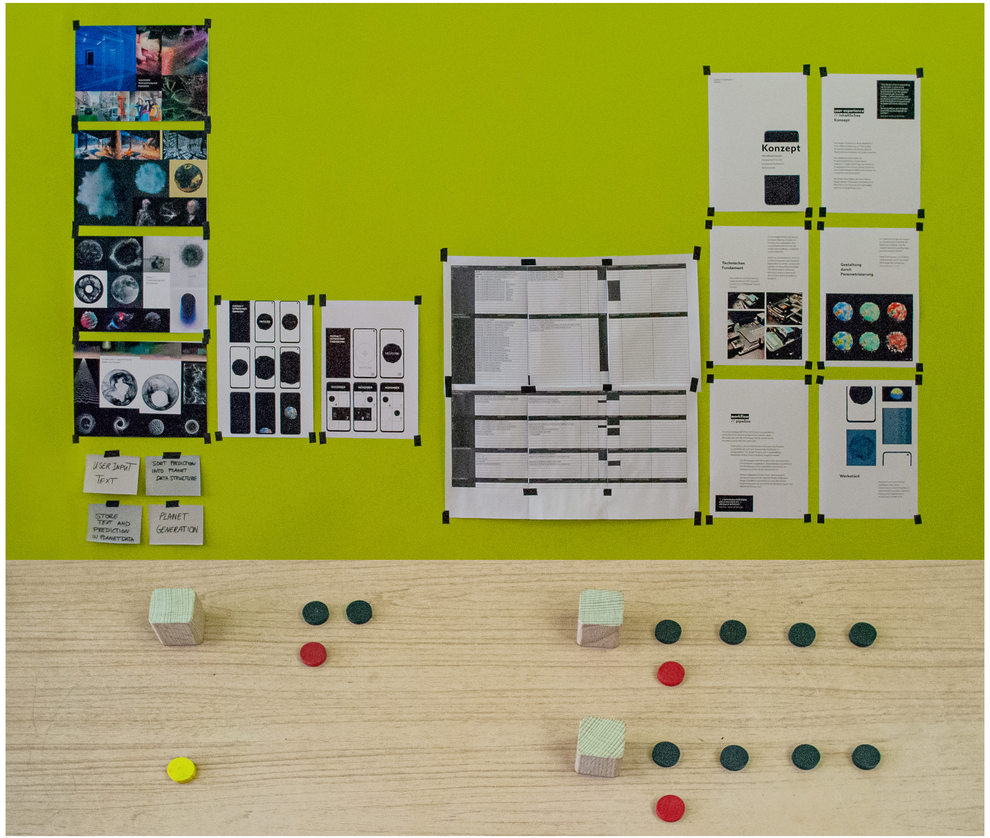


Abb. 3: Arbeitssituation 01

Abb. 4: Analoges Prototyping für UI

IDEATION

Der Ideationprozess für dieses Projekt lief in mehreren Stufen ab. Um sich mit dem Feld der Mensch-Maschine Kommunikation auseinander zu setzen ist es vorab notwendig den technischen Hintergrund dieses Themenbereichs zu untersuchen und eine Grundlage für die folgende Ideenfindung zu definieren. Diese Grundlage bildet hier die Betrachtung zeitgemäßer, relevanter Deep Learning Modelle, die das gesamtgesellschaftliche Verhältnis im Umgang mit Technologie nachhaltig beeinflussen. Der erste Schritt ist also, einen Maschine Learning Bereich auszumachen, der inhärent interessant für interaktionsbasierte Medienprodukte ist.

Anschließend sind mögliche Anwendungsfelder und interessante Interaktionsmuster zu identifizieren, die die Grundlage für die Entwicklung von Prototypen bilden und als Vorzeigeprojekte für die kreative Erschließung jener innovativen Technologien dienen können. Hier sind sowohl Einbindungen in gegebene Produktionsabläufe digitaler Spiele, als auch eigenständige Erlebniskonzepte denkbar. Es gilt abzuwägen welche Ansätze im Rahmen dieser Bachelorarbeit als Proof of Concept umgesetzt werden können und welche Ideen als Ausblick für die weitere Beschäftigung mit der Thematik konzeptionell festgehalten werden.

Abschließend sollte darüber nachgedacht werden, wie die Ergebnisse dieses explorativen Prozesses zugänglich gemacht werden können. Dies geschieht zum einen mit dem Verfassen dieses Projektbuches, zum anderen über die Offenlegung und Dokumentation wiederverwendbarer Code-Elemente oder — wie im Beratungsgespräch durch meinem betreuenden Professor angeregt — durch die Entwicklung und Bereitstellung einer Programmierschnittstelle, die Folgeprojekte anderer Game Designer*innen und Entwickler*innen in der technischen Umsetzung unterstützen kann.

TECHNOLOGISCHE INNOVATION

Der Umstand, dass die Entwicklung hochmoderner Algorithmen mehr und mehr den Funktionalitäten des menschlichen Gehirns nachempfunden sind, hat einen nicht zu unterschätzenden Einfluss auf unser rationales wie emotionales Verhältnis zur Technologie selbst. Computer Vision, Speech Recognition, Object Detection und Emotional Understanding eröffnen neue unmittelbare Kommunikationswege zwischen dem Anlogen, unserer physischen Lebens- und Erfahrungswelt und dem Digitalen, den international vernetzten Informationskanälen des Internets und des analytischen Datenverarbeitungsvermögens unzähliger vernetzter Prozessoren. Ob durch Gesichtserkennung, visuelle Augmentierung von Kamerabildern oder selbstfahrende Autos — der gesellschaftliche Einfluss interpretativer Rechensysteme nimmt rapide zu. So müssen sich besonders im Bereich des Interaktions- und Game Designs immer wieder die Fragen gestellt werden; “Welche Problemdefinitionen motivieren unsere Projekte?”, “Wie transparent nutzen wir neuartige Technologien?” und “Welche gesellschaftlichen Prozesse werden durch Projekte — beabsichtigt oder unbeabsichtigt — beeinflusst?”

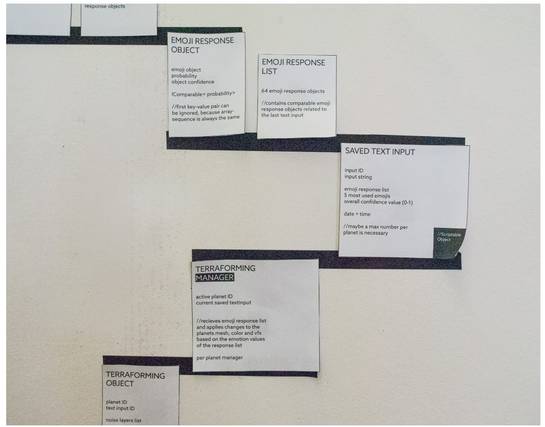


Abb. 5: Analoges Prototyping für Klassendiagramm 01

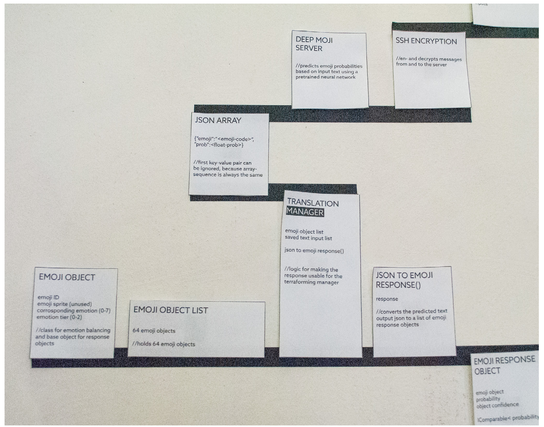


Abb. 5: Analoges Prototyping für Klassendiagramm 02

Game Designer*innen entwerfen interaktionsbasierte Erfahrungen in erster Line durch die systemische Konfiguration von möglichen Handlungsabläufen in erlernbaren Regelwerken, die oft durch Gameloop-Iterationen von Spieler*innen optimiert werden können. Diese Handlungsabläufe lassen sich in parametrisierbare Inputs unterteilen, die den Regeln des Spiels entsprechend interpretiert werden, neuen Output generieren und somit neue Inputreaktionen der Spieler*innen provozieren.

Diesen technischen Werkzeugkatalog parametrisierbarer Inputs zu erweitern, wirkt sich also unmittelbar auf die Gestaltung neuer Regelwerke und somit auf neue Spielkonzepte aus. Einerseits kann dies durch den Ausbau der Schnittstelle zur Spieler*in geschehen, indem neue Hardware erprobt und nutzbar gemacht wird — hierzu zählen alle möglichen Sensoriken, wie beispielsweise Gyroskope, Kameras, Mikrofone oder Knopf und Joystick Anordnungen — andererseits kann dies durch softwarebasierte Verbesserung des Verständnisses der Inputwerte stattfinden, wozu unter anderem das breite Feld des Maschine Learning gehört. Im technisch grundlegenden Teil der Ideation habe ich mich gezielt mit der softwareseitigen Erweiterung des Inputverständnisses beschäftigt und mich letztlich — der systemischen Struktur von Neuralen Netzwerken folgend — dem Bereich gewidmet der meiner Auffassung nach das Menschliche im Maschinellen aktuell am lebhaftesten nachbildet: Emotional Recognition und Emotional Understanding.

Für die Evaluation der Anwendungsfelder und der Ausrichtung meiner Prototypen war die Ideenfindungsphase bis zum Projektabschluss nie komplett beendet. Der explorative Charakter dieser Arbeit führte an vielen Stellen zu neuen Entwürfen für die Präsentation. Diese reichten von räumlichen Installationen, über Gesprächssimulationen mit einem Chatbot bis hin zu einem interaktiven Tagebuch. Die verschiedenen Ansätze sind in diesem Projektbuch festgehalten, letztlich habe ich meine Arbeit auf die Entwicklung einer API konzentriert, die als konkretes, im Rahmen dieser Bachelorarbeit finalisierbares Werkstück künftig die Entwicklung zahlreicher solcher Projektideen in der Unity Engine ermöglichen wird. Für die Entwicklung dieser API sind weitere Ideationschritte notwendig gewesen, die sich nun zum einen mit den speziellen Anforderungen an, für Entwickler*innen produzierte, Inhalte beschäftigte und zum anderen die Gestaltung der Systemarchitektur so erweiterbar und wiederverwendbar wie möglich konzipieren sollte.

Die Entwicklung vom theoretischen Interesse über die prototypische Erprobung hin zum abschließenden Produkt findet sich auch in der Workflowgestaltung wieder, da zwar zu Beginn des Projektes noch nicht klar definiert werden konnte, welches Produkt am Ende steht, das stufenhafte Ausprobieren und Neuorientieren aber, aufgrund der Komplexität der gewählten Thematik, absehbar war.

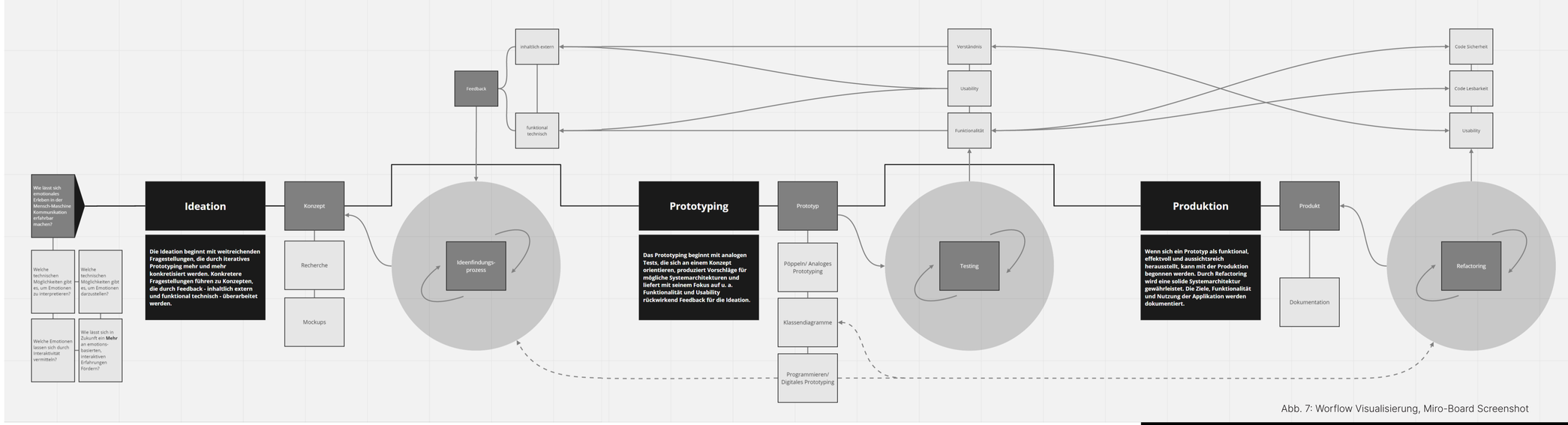


Abb. 7: Worflow Visualisierung, Miro-Board Screenshot

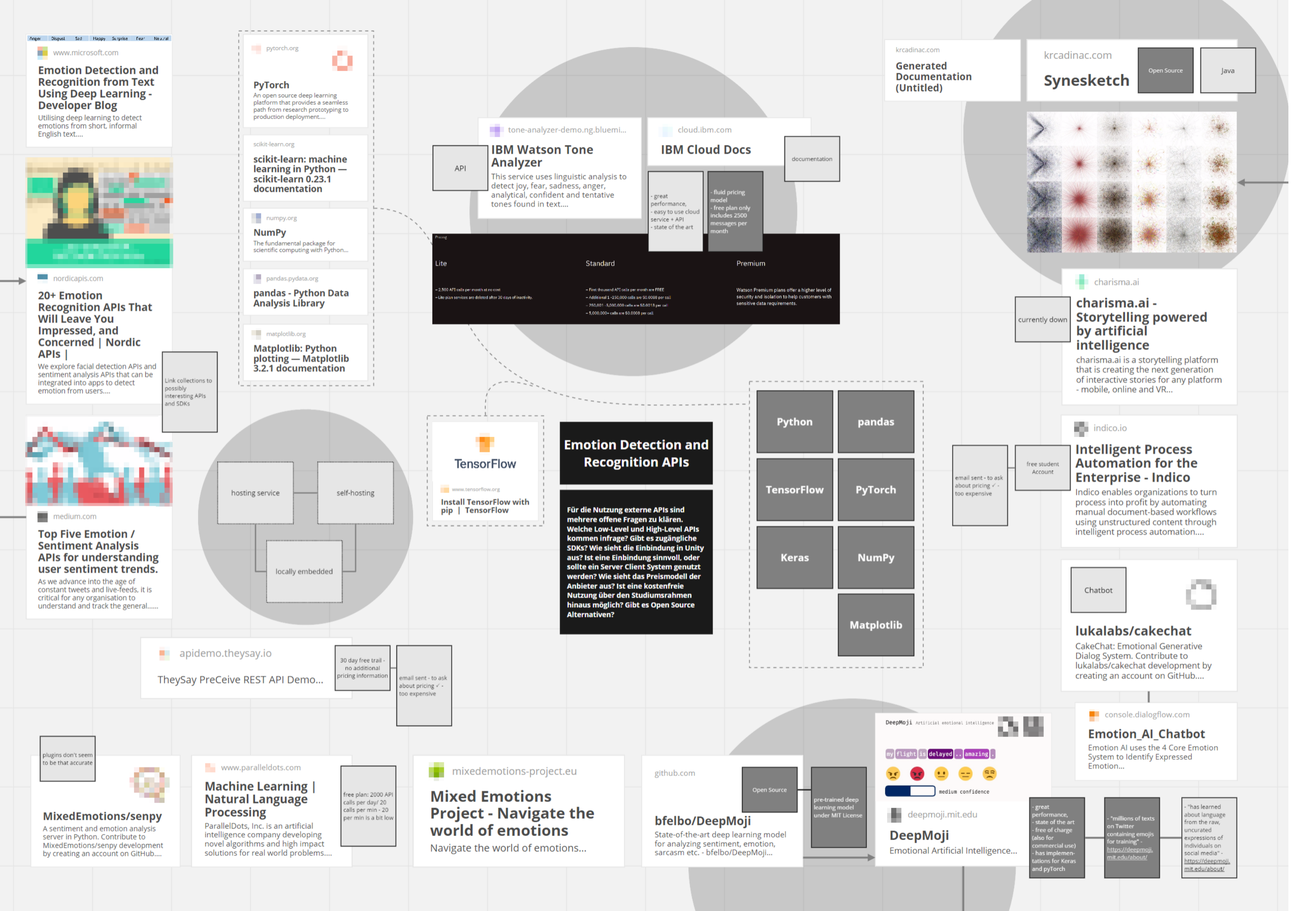
WORKFLOW

Im oben stehenden Diagramm ist die Workflowgestaltung für das Designprojekt visualisiert. Von links nach rechts sind dort mehrere Prozessbeschreibungen und Iterationsschleifen aufgeführt und beschrieben. Am Anfang stehen mehrere theoretische Fragestellungen, die über die Ideation in ein Konzept einfließen. Dieses Konzept bildet das Fundament für Prototyping, das wiederum durch gezielte Testings Feedback für den Ideenfindungsprozess generiert. So entstehen neue Ansätze, die wieder prototypisch getestet werden, bis der Fokus auf die konkrete Produktion verlagert wird. Die Produktion strukturiert wiederum durch Refactoring und setzt die gesammelten Erkenntnisse der vorgeschalteten Prozesse in einem Produkt, dem Werkstück dieser Bachelorarbeit, um.

Die Bereiche der Ideation und des Prototyping sind im Laufe der Arbeit für verschiedene Ansätze durchlaufen worden, haben aber in jeder Iteration Einfluss auf das finale Werkstück genommen. Die Überarbeitung von Konzepten, Ansammlung von Ideen, Durchführung von Playtest, Erarbeitung von Klassendiagrammen und Code-Elementen ist ein wesentlicher Bestandteil der Bedarfs- und Machbarkeitsbeurteilung für die im Anschluss entwickelte API gewesen. Genauere Aufschlüsselungen der einzelnen Prozesse, sowie konkrete Code-Beispiele zum Thema Sicherheit und Lesbarkeit finden sich in den entsprechenden Kapiteln. Eine ausführliche Dokumentation, die die Nutzung und Funktion des Werkstücks für nachfolgende Entwickler erläutert, ist diesem Projektbuch angefügt.

AGILE ZIELSETZUNG

Durch den experimentellen Charakter dieses Projekts erklärt sich der prototyporientierte Workflow. Emotional Understanding wird aktuell noch kaum für spielerische Erfahrungen als Design-Instrument genutzt. Um die Brauchbarkeit für verschiedene Anwendungsbereiche zu evaluieren und, wenn möglich, mit dieser Arbeit zugänglich zu machen, ist ein sondierender Workflow mit engmaschigen Iterationen nötig. Selbstverständlich kann dies hier nur exemplarisch geschehen, da eine erschöpfende Untersuchung, neben sozialwissenschaftlichen Studien, auch weitreichende repräsentative Playtests mit unterschiedlichen Zielgruppen berücksichtigen müsste. Die Zielsetzung hat sich also immer wieder an den Ergebnissen der Entwicklungsprozesse orientiert, die letztlich als Grundlage für das finale Werkstück dienten.



EXTERNE APIs

Die Suche nach einer Open-Source Lösung für Emotional Understanding Algorithmen stellte sich als relativ schwierig heraus. Zu den Marktführern gelten neben Google und IBM diverse mittelständische Unternehmen, die ihre Lösungen meist als Abo-Modelle anbieten oder die Kosten anhand der API Zugriffszahlen berechnen.

Da sich das Designprojekt SELF-INFLICTED EMPATHY nicht nur auf einer theoretischen Ebene mit Möglichkeiten von KI-gestützter Emotionsanalyse beschäftigen, sondern darüber hinaus praktische Anwendung in Folgeprojekten finden soll, bieten solche Schnittstellen keine nachhaltige Lösung für das technische Fundament dieser Arbeit.

Unity3D ist die in diesem Projekt verwendete Game-Engine. Aufgrund der jahrelangen Erfahrung die ich mit dieser Entwicklungsumgebung während meines Studiums sammeln konnte, ist es an dieser Stelle entscheidend gewesen, eine Plattform zu wählen, die eine Einbindung in meine vorhandenen Kompetenzbereiche ermöglicht.

WATSON TONE ANALYZER

IBM beschreibt seinen Tone Analyzer Service wie folgt: "Der IBM Watson™ Tone Analyzer-Service verwendet linguistische Analyse zum Erkennen von emotionalen und sprachlichen Tonfällen in geschriebenem Text." (IBM o.V. 2019) Mit eintausend kostenfreien API Aufrufen pro Monat ist IBM's Watson Tone Analyzer zwar ein zugängliches Tool für Prototyping- und Testfälle, darüber hinaus, ist eine Anwendungsskalierung aber nur mit Nutzer*innen bezogener Monetarisierung denkbar, da jeder anschließende API Aufruf von Entwickler*innen bezahlt werden muss.

SYNESKETCH

Synesketch ist eine via Open Source verfügbare Abschlussarbeit von der University of Belgrade. Es handelt sich um eine in Java geschriebene Engine, die Emotionen in Texten erkennt und visualisiert. Die auf der Website präsentierten Visualisierungen sind erkennbar der künstlerischen Expression gewidmet. (vgl. Krcadinac, Pasquier 2020) So interessant dieses Projekt ist, so spärlich dokumentiert und unzugänglich ist es leider auch. Darüber hinaus bin ich mit Java wenig vertraut, was die Einarbeitung nicht leichter macht.

DEEP MOJI

DeepMoji ist ein auf 1.2 Milliarden Twitter-Texten trainiertes Deep Learning Modell, welches von Forscher*innen des Media Lab - Massachusetts Institute of Technology, College of Computer and Information Science - Northeastern University, Department of Computer Science - University of Copenhagen und dem DTU Compute - Technical University of Denmark kollaborativ entwickelt und unter der MIT Lizenz veröffentlicht wurde. DeepMoji hat eine überragende Vorhersagegenauigkeit, die in den Bereichen Ironie und Sarkasmus selbst die erfolgreichsten pay-per-APIcall oder Abolösungen übertrifft.

Abb. 8: Emotional Understanding API Recherche, Miro-Board Screenshot

Abb. 9: DeepMoji Emoji-Klassen

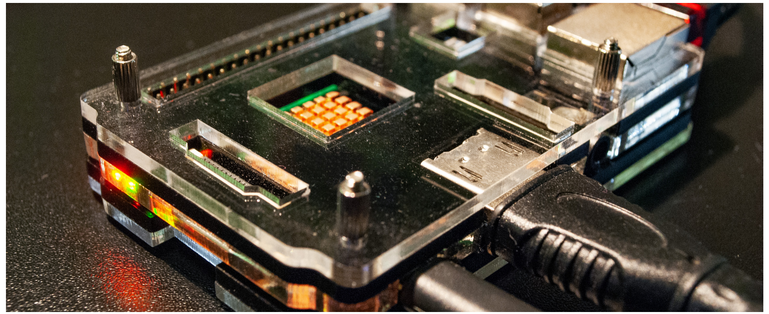
Da DeepMoji die beeindruckendsten Ergebnisse liefert und kostenlos ohne substanzielle Einschränkungen zur Weiterverarbeitung und zur kommerziellen Nutzung zur Verfügung steht, habe ich mich entschlossen dieses Framework als die Grundlage meiner Bachelorarbeit heranzuziehen. Da das Modell bereits richtig gewichtet ist und via öffentlichem Git Repository zur Verfügung steht, galt es eine Lösung zu finden, die das vortrainierte Modell ohne fundamentale Änderungen in mein Projekt implementiert und adressierbar macht.

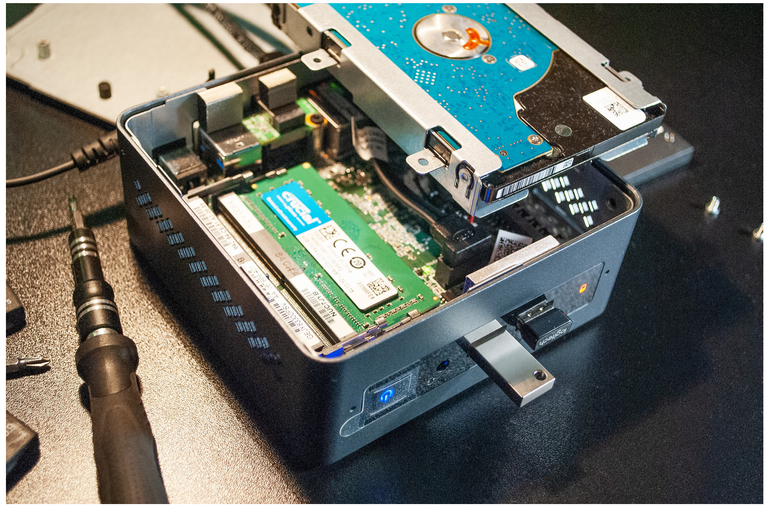
Auf der linken Seite sind die 64 Emojis zu sehen, auf die das DeepMoji Modell trainiert ist und für die es Vorhersagen auf Basis von Text treffen kann. Die inhaltlichen Dopplungen mancher Emojis rühren wohl daher, dass die Auswahl auf der Auswertung der Trainingsdaten und somit der Häufigkeit der Benutzung und der Eindeutigkeit der Aussagekraft liegt.

Auf der rechten Seite ist eine Illustration zu sehen, die dem Paper "Using millions of emoji occurrences to learn any-domain representationsfor detecting sentiment, emotion and sarcasm" nachempfunden ist und die verschiedenen Layer des Models veranschaulicht. (vgl. Felbo et al. 2017 S.2)

Den Anfang macht ein standard Embedding Layer, der den eingegebenen Text via natural language processing (NLP) in verrechenbare Vektoren übersetzt. Es schließen sich zwei long short-term memory (LSTM) Layer an, die den eingegebenen Text vorwärts und rückwärts durchlaufen, um den Kontext einzelner Worte richtig einschätzen zu können. Der nächste Layer ist ein Attention Netzwerk, dass die Relevanz einzelner Worte für die finale Vorhersage auf Basis des zuvor ermittelten Textaufbau-Verständnisses beurteilt. Abschließend folgt ein standard Softmax Layer, der die Ergebnisse auf die vordefinierten Klassen - in diesem Fall die 64 verschiedenen Emojis - aufteilt.

Abb. 10: Illustration des DeepMoji Modells, Miro-Board Screenshot





TECHNISCHE GRUNDLAGEN

Für das Deployment des DeepMoji Modells gibt es zwei grundlegende Möglichkeiten: Eine native Einbindung in die App Struktur und ein Server-Client System. Beide Möglichkeiten bieten Vor- und Nachteile für die weitere Nutzung des Projektes. Während eine native Einbindung keine bestehende Internetverbindung benötigt, um Voraussagen zu treffen, ist eine Server-Client Lösung schlanker und entkoppelt das Deep Learning Modell von der Interpretationslogik der Applikation.

Im Hinblick auf die Zugänglichkeit der Ergebnisse dieser Bachelorarbeit und der potentiellen Nutzung für Folgeprojekte sorgt eine solche Entkoppelung ebenfalls für eine Reduzierung der Abhängigkeiten und vereinfacht eventuelle Refactorings. Es ist außerdem anzumerken, dass eine Server-Client Lösung für den Rahmen einer räumlich gebundenen Installation auch im lokalen Netzwerk aufgesetzt werden kann, ohne dass eine Internetverbindung benötigt wird.

Grundsätzlich war also bereits zu Beginn eine Präferenz zu einer Server-Client Struktur gegeben. Gleichwohl galt es den möglichen Implementierungsvorgang eines trainierten Deep Learning Modells in Unity zu erproben, um mögliche unvorhergesehene Vorteile dieses Ansatzes zu ermitteln und anschließend eine durch neue Erkenntnisse geschulte finale Entscheidung treffen zu können.

Abb. 11: Hardwaresetup Raspberry Pi 2

Abb. 12: Hardwaresetup Intel NUC

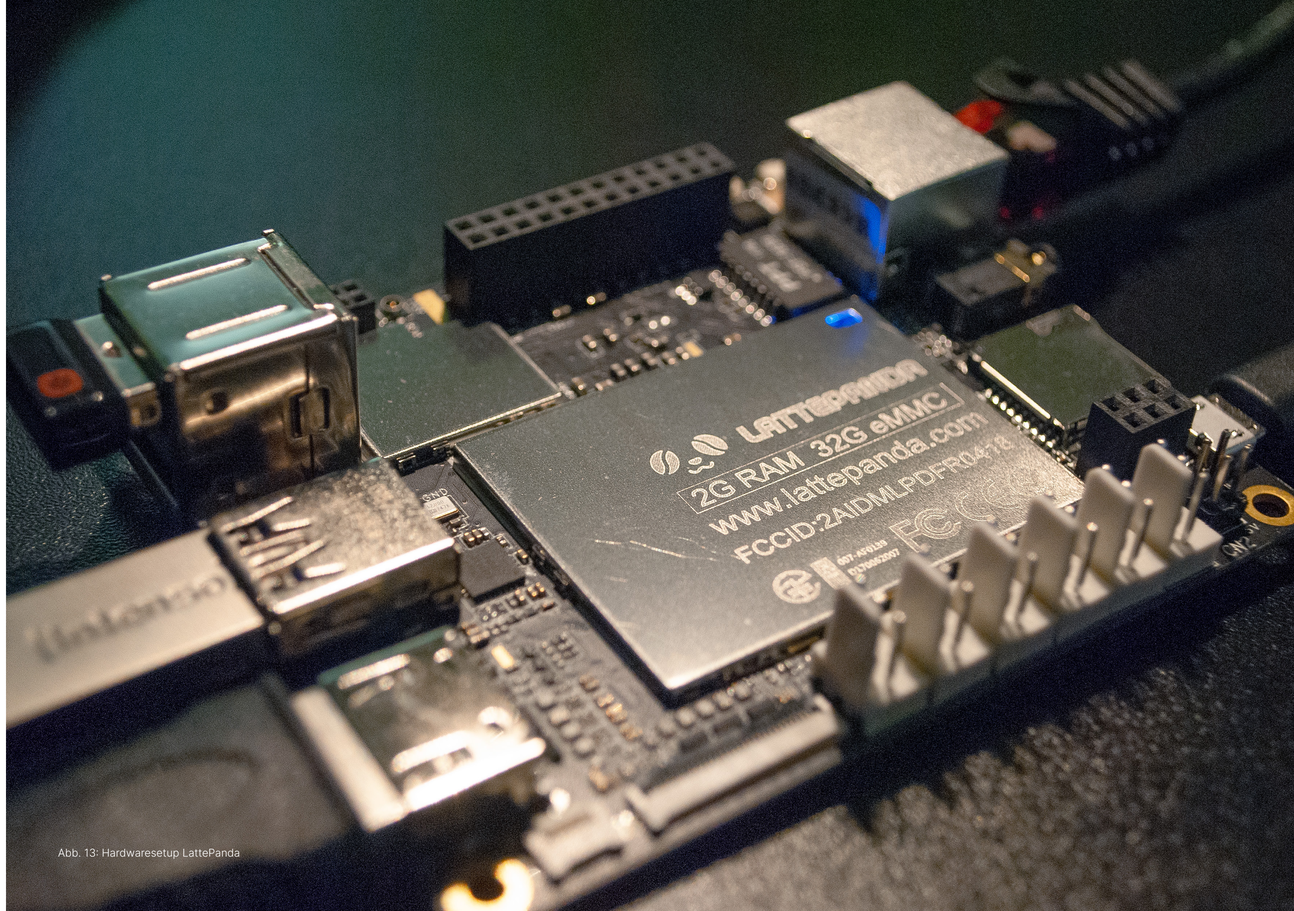


Abb. 13: Hardwaresetup LattePanda

DEPLOYMENT Für die lokale Einbindung von Machine Learning Modellen hat Unity im September 2017 das Open Source Projekt Unity Machine Learning Agents Toolkit gestartet, das laufend weiter entwickelt wird. Der Fokus dieses Projektes liegt auf dem Training intelligenter Agenten durch Reinforcement Learning und einer auf Tensor Flow basierenden Implementierung verschiedener Trainingsalgorithmen. (vgl. Berges et al. 2020) Der Output von ML-Agents ist somit vor Allem dazu gedacht NPC Verhalten zu beeinflussen und automatisierte Testings durchzuführen. Zum Zeitpunkt meiner Kompatibilitätstests unterstütze Unity ML-Agents aber keine LSTM-Layer. DeepMoji nutzt diese Layer, um ein umfassendes Kontextverständnis der eingegebenen Texte zu erhalten, was für die Funktionalität des Modells unabdingbar ist. Dieser Umstand alleine schließt die Nutzung des von Unity bereitgestellten Toolkits bereits aus.

Darüber hinaus habe ich mit verschiedenen anderen Plugins wie Tensor Flow Sharp experimentiert und habe die Tests aber aus einem übergeordneten Grund eingestellt: Die Größe des Modells. DeepMoji belegt inklusive Tensor Flow Implementierung und mit allen relevanten Gewichten (weights) über 2.5 Gigabyte an Speicherplatz. Für eine prototypische Applikation mag das akzeptabel sein, aber mit dem Blick auf ein fertiges Produkt, besonders, wenn es eine Mobile App sein sollte, ist dieser Datenumfang viel zu hoch. Somit war klar, dass ein Server-Client basierte Ansatz eine skalierbarere Lösung ist, die auch die Produktion von leichtgewichtigen Applikationen und die Augmentierung vorhandener Systeme besser unterstützt.

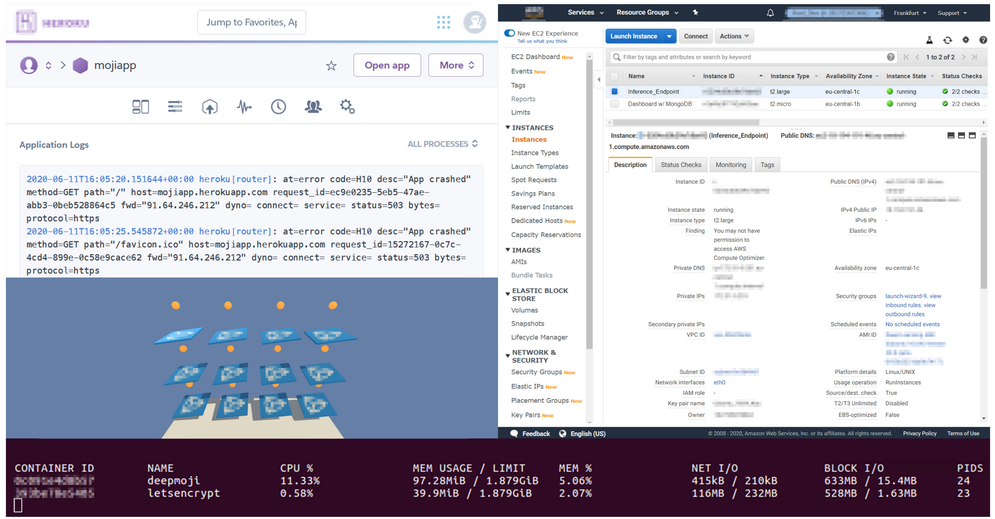


Abb. 14: Mögliche Deployment Plattformen Collage, Browser Screenshots

Die nächste zu klärende Frage war nun, ob für diese Lösung ein Online Hosting-Service genutzt werden sollte, oder ob das Deployment auf einem eigenen Server sinnvoller ist. Ich habe mich diesbezüglich mit den Anbietern Heroku und Amazon Web Services (AWS) beschäftigt. Da AWS ein ähnliches Bezahlmodell benutzt, wie ich es bereits zuvor bei IBM gesehen habe, war dieser Service, aufgrund der geringen Skalierbarkeit über ein Universitätsprojekt hinaus, relativ uninteressant. Heroku hingegen bietet ein kostenfreies Hosting einer Webapp an, das nicht durch die Anzahl der Anfragen, sondern der Rechenzeit der Applikation limitiert ist. Trotz dieser Einschränkung habe ich den Heroku Service ausprobiert, konnte in den plattformspezifischen dyno Containern das Modell aber nicht zum Laufen bringen.

Letztendlich fiel der Entschluss darauf, dass Modell auf einem eigenen Server zu hosten. Diese Entscheidung machte das Designprojekt nicht nur reproduzierbarer und skalierbarer, sie gab mir auch mehr Kontrolle über die Laufzeitumgebung des Modells. Folglich konnte ich für das Aufsetzen des Servers frei aus gut dokumentierten Lösungen wählen, was unter dem Gesichtspunkt, dass ich mich auf persönlich-fachlichem Neuland bewegte, ein großes Plus war. Für die kompakte und sichere Bereitstellung des Modells auf meinem eigenen Webserver habe ich mich für Docker entschieden, ein Containerservice, der sich wachsender Beliebtheit unter Webapp Entwickler*innen erfreut. Auf den kommenden Seiten ist die Miro Dokumentation meiner ersten lokalen Versuche unter Windows 10 aufgeführt.

DeepMoji Deployment

Abb. 15: DeepMoji Deployment Recherche und eigene Anleitung, Miro-Board Screenshot

Dieses Board zeigt exemplarisch den Deployment-Prozess eines DeepMoji Docker Containers in einer Docker for Windows Umgebung. Für das erfolgreiche Aufsetzen eines Docker Containers in einer Linuxumgebung, auf einem Einplatinencomputer oder über einen online Hostingservice sind weitere Schritte notwendig, deren Evaluation sich an die Erarbeitung dieser ersten Tests angeschlossen hat.

Die folgenden Tests beruhen auf dem Container aus diesem Repository:

github.com

nolis-llc/DeepMoji-docker

A docker container for running DeepMoji as a service - nolis-llc/DeepMoji-docker...

install docker for windows (accout required)

navigate into repo and build the image from the Dockerfile by typing:

`docker build .`

docker build .

you can see all existing images by typing:

docker images

```
C:\Users\Caspar\Desktop\Test_Mojis\DeepMoji-docker-master>docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
<none>              <none>              7ce4d17b5c2d        26 seconds ago      2.83GB
<none>              <none>              2298b1399a0c        3 hours ago         2.83GB
python              2.7.15-stretch      afc94ac1e19b        3 months ago        925MB
```

Python Version muss manuell eingeflugt werden

Open server.py and add the following line infront of everything else:

`#!/usr/local/bin/python2.7`

github.com

0.15.0 causes OSError: [Errno 8] Exec format error: in Docker for Windows · Issue #1482 · pallets/werkzeug

The new 0.15.0 does not run in Docker for Windows. Have not tried Docker on other platforms. Minimal example with Flask. app.py from flask import Flask app = Flask(__name__) @app.route('/').....

docker run -p 80:80 --name test2 7ce4d17b5c2d

port

container name

image ID

docker stop [containername]

docker run -p 80:80

create and run container by:

to give access and portst to running container add the following while executing run command:

docker ps

docker stats

monitor the running container by executing:

deepMoji docker debugger:

```
* Debugger is active!
* Debugger PIN: [redacted]
[redacted] - - [redacted] "GET / HTTP/1.1" 405 -
[redacted] - - [redacted] "GET /favicon.ico HTTP/1.1" 404 -
[redacted] - - [redacted] "GET / HTTP/1.1" 405 -
[redacted] - - [redacted] "POST / HTTP/1.1" 400 -
[redacted] - - [redacted] "POST / HTTP/1.1" 400 -
[redacted] - - [redacted] "POST / HTTP/1.1" 400 -
[redacted] - - [redacted] "POST / HTTP/1.1" 200 -
```

docker docs

docs.docker.com

Networking features in Docker Desktop for Windows

Networking...

docs.docker.com

Install Docker Desktop for Windows

How to install Docker Desktop for Windows...

www.python.org

Python Release Python 2.7.15

The official home of the Python Programming Language...

install Python 2.7 && pip

virtualenv.pypa.io

Reference Guide — virtualenv 16.6.1 documentation

install virtualenv

pip.pypa.io

Installation — pip 19.1.1 documentation

default PATH:

www.howtogeek.com

How to Edit Your System PATH for Easy Command Line Access in Windows

Have you ever wondered why you can't just type ipconfig into a command prompt from anywhere, but when you want to use a command line program you downloaded you have to be in that directory first? Here's how to fix that, using the Windows System PATH...

gnuwin32.sourceforge.net

wget for Windows

wget (whatisit)...

install wget && configure system PATH

C:\Program Files (x86)\GnuWin32\bin

```
Administrator: Command Prompt - docker stats
CONTAINER ID        NAME                CPU %               MEM USAGE / LIMIT     MEM %               NET I/O             BLOCK I/O           PIDS
[redacted]          test3               2.58%               834.8MiB / 1.934GiB   42.15%              8.55kB / 14.1kB     31MB / 41kB         24
```

```
{"emoji":[[{"emoji": "ÆfÿÀ", "prob": 0.00309789879
247546196},{"emoji": "ÆfÿÆ", "prob": 0.00025202688
993886113},{"emoji": "ÆfÿÑ", "prob": 0.00302823632
955551147},{"emoji": "Æfÿì", "prob": 0.00341931707
225739956},{"emoji": "ÆfÿÈ", "prob": 0.04443262144
923210144},{"emoji": "ÆfÿÖ", "prob": 0.00028928526
444360614},{"emoji": "ÆfæÌ", "prob": 0.02067184261
977672577},{"emoji": "Æfÿi", "prob": 0.02849609032
273292542},{"emoji": "Ì¤ÑÑ", "prob": 0.063473016023
63586426},{"emoji": "ÆfÿÄ", "prob": 0.002998394425
95839500},{"emoji": "Æfÿü", "prob": 0.016473239287
73403168},{"emoji": "ÆfÃ[redacted]", "prob": 0.229309156537
05596924},{"emoji": "Æfÿ[redacted]", "prob": 0.000818981672
63716459},{"emoji": "Æfæ»", "prob": 0.003258602460
84630489},{"emoji": "Æfÿ[redacted]", "prob": 0.000394015194
61534917},{"emoji": "Æfÿï", "prob": 0.010622423142
19474792},{"emoji": "Æfÿè", "prob": 0.024476898834
10930634},{"emoji": "ÆfÖï", "prob": 0.017720922827
72064209},{"emoji": "Æfæò", "prob": 0.024834981188
17806244},{"emoji": "Æfÿæ", "prob": 0.000259769789
40889239},{"emoji": "Æfÿà", "prob": 0.001640177564
69547749},{"emoji": "ÆfÖÄ", "prob": 0.002723870566
11478329},{"emoji": "Æfÿò", "prob": 0.000133495690
53389132},{"emoji": "Æfÿÿ", "prob": 0.008655810728
66916656},{"emoji": "ÆfÕÑ¬Ä", "prob": 0.0557971820
2352523804},{"emoji": "ÆfÿÉ", "prob": 0.0002172876
9934270531},{"emoji": "Æfæü", "prob": 0.0034313364
8857474327},{"emoji": "Æfÿ¾", "prob": 0.0002651144
4593779743},{"emoji": "ÆfÖê", "prob": 0.0053617451
3399600983},{"emoji": "Æfÿú", "prob": 0.0025028034
5790088177},{"emoji": "ÆfÃî", "prob": 0.007771296891
231514931},{"emoji": "ÆfÿÄ", "prob": 0.00829022191
464900970},{"emoji": "Æfÿì", "prob": 0.00022524208
179675043},{"emoji": "Æfæì", "prob": 0.02422750927
507877350},{"emoji": "ÆfÿÑ", "prob": 0.00091135234
106332064},{"emoji": "Æfÿ¬", "prob": 0.00045810508
891008794},{"emoji": "Æfÿì", "prob": 0.00765120843
425393105},{"emoji": "ÆfÿÑ", "prob": 0.00035276779
090054333},{"emoji": "ÆfÑÙ", "prob": 0.00070891121
868044138},{"emoji": "Æfÿ¸", "prob": 0.00018222874
496132135},{"emoji": "ÆfæÀ", "prob": 0.01597132720
053195953},{"emoji": "ÆfæC", "prob": 0.00065898627
508431673},{"emoji": "ÆfÖ¼", "prob": 0.00192414922
639727592},{"emoji": "Æfÿú", "prob": 0.00039297118
200920522},{"emoji": "Æfÿê", "prob": 0.00258683552
965521813},{"emoji": "Æfÿõ", "prob": 0.00022380618
611350656},{"emoji": "ÆfÆõ", "prob": 0.00066862307
721748948},{"emoji": "ÆfÆÙ", "prob": 0.05896438285
708427429},{"emoji": "ÆfÃº", "prob": 0.11035272479
057312012},{"emoji": "ÆfÖè", "prob": 0.00556388869
881629944},{"emoji": "Æfÿë", "prob": 0.00319417985
156178474},{"emoji": "ÆfÆC", "prob": 0.00115089095
197618008},{"emoji": "Æfÿü", "prob": 0.00029360930
784605443},{"emoji": "Æfÿà", "prob": 0.03643383085
727691650},{"emoji": "Æfÿ£", "prob": 0.00771251413
971185684},{"emoji": "Æfÿà", "prob": 0.00028242811
094969511},{"emoji": "ÆfÖä", "prob": 0.00063304230
570793152},{"emoji": "Æfæ¬", "prob": 0.00266611506
231129169},{"emoji": "Æfæè", "prob": 0.00295770517
550408840},{"emoji": "Æfæ£", "prob": 0.04436102882
027626038},{"emoji": "ÆfÆÙ", "prob": 0.01650088894
174861908},{"emoji": "ÆfÆÖ", "prob": 0.04509197548
031806946},{"emoji": "Æfÿ¥", "prob": 0.00517955329
269170761},{"emoji": "Æfæ¿", "prob": 0.006399080622
94125557}],[{"emoji": "ÆfÿÀ", "prob": 0.0024313561
```

get curl by:

installing git for windows

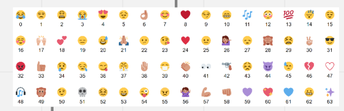
get it here

Download Git for Windows, and it will install CURL along with it. You can find it installed under C:\Program Files\Git\mingw64\bin\. Add it to your Windows path, and you will be able to execute it from anywhere.

Press the start button, and type system path. It will display an option to edit System Variables. Click on Environmental Variables, and you will be able to add the path as mentioned above into the system path.

windows command promt (CMD) is unable to use single quotation marks. insert doubble quotation marks instead and switch former double quotation marks with \"
so just put a backslash in front of them

curl -X POST http://127.0.0.1 -H "Cache-Control: no-cache" -H "Content-Type: application/json" -H "Postman-Token: [redacted]" -d '{\"sentences\":[\"I love this band!\", \"I hate this album.\"]}'



you can stop a running container by typing:

github.com

deepMoji Python3

zzsza/DeepMoji-Python3

DeepMoji Python3 (Original Repo : https://github.com/bfelbo/DeepMoji) - zzsza/DeepMoji-Python3...

TensorFlow

www.tensorflow.org

Install TensorFlow with pip | TensorFlow

stackabuse.com

Dockerizing Python Applications

Introduction Docker is a widely accepted and used tool by leading IT companies to build, manage and secure their applications. Containers, like Docker, allow developers to isolate and run multiple applications on a single operating system, rather than de...

www.dropbox.c...

deepMoji weights direct download

deepmoji_weights.hdf5

Shared with Dropbox...

github.com

cURL POST command does not work on Windows Command Prompt because single quotes are used · Issue #11 · spring-guides/gs-accessing-data-rest

Windows Command Prompt treats quotes differently when compared to the Unix shell and this causes an error when the developer tries to create a new person using cURL to post data. > curl -X POST .....

docker docs

docs.docker.com

Get started with Docker for Windows

Getting Started...


```
A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default
                           "C:\\Users\\Caspar\\.docker")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level
                           ("debug"|"info"|"warn"|"error"|"fatal")
                           (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default
                           "C:\\Users\\Caspar\\.docker\\ca.pem")
      --tlscert string     Path to TLS certificate file (default
                           "C:\\Users\\Caspar\\.docker\\cert.pem")
      --tlskey string      Path to TLS key file (default
                           "C:\\Users\\Caspar\\.docker\\key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  builder     Manage builds
  config      Manage Docker configs
  container   Manage containers
  image       Manage images
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes

Commands:
  attach      Attach local standard input, output, and error streams to a running container
  build       Build an image from a Dockerfile
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  events      Get real time events from the server
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  history     Show the history of an image
  images      List images
  import      Import the contents from a tarball to create a filesystem image
  info        Display system-wide information
  inspect     Return low-level information on Docker objects
  kill        Kill one or more running containers
  load        Load an image from a tar archive or STDIN
  login       Log in to a Docker registry
  logout      Log out from a Docker registry
  logs        Fetch the logs of a container
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  ps          List containers
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  rmi         Remove one or more images
  run         Run a command in a new container
  save        Save one or more images to a tar archive (streamed to STDOUT by default)
  search      Search the Docker Hub for images
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  version     Show the Docker version information
  wait        Block until one or more containers stop, then print their exit codes

Run 'docker COMMAND --help' for more information on a command.
```

BETRIEBSSYSTEM UND HARDWARE

Nach den ersten lokalen Versuchen auf meinem PC habe ich verschiedene Einplatinencomputer als Server in Betracht gezogen. Das erste Versuchsobjekt war ein Raspberry Pi 2. Es stellte sich schnell heraus, dass die, in dem von mir benutzten Docker Container referenzierte, Tensor Flow Version die Prozessorarchitektur des Pi nicht unterstützte. Mittlerweile habe ich herausgefunden, wie ich dieses Problem hätte lösen können, zu dem Zeitpunkt in der Entwicklung befand ich mich aber noch am Anfang des Lernprozesses. Nach einer kurzen Testiteration auf einem Intel NUC entschloss ich mir einen LattePanda zuzulegen, der mit einer kompatiblen CPU ausgestattet ist.

Aufgrund des unnötig großen Overheads von Windows 10, entschied ich mich dazu Linux als Betriebssystem zu nutzen. Ubuntu schien mir eine gute Version zu sein, die für Erstnutzer ein leichtgewichtiges, aber übersichtliches und funktionales User Interface bietet, dass es ermöglicht den Server nicht nur über remote Commands einzurichten. Da zu dem Zeitpunkt nicht alle Ubuntu Versionen stabil auf dem LattePanda liefen, entschloss ich mich dazu die Ubuntu 16.04 Long-Term Support (LTS) Version zu nutzen.

Erste Tests auf dem LattePanda brachten allerdings Probleme mit sich. Trotz der grundsätzlichen Kompatibilität der Systemarchitektur und des Intelchipsatzes konnte Tensor Flow aufgrund Konflikten mit dem Prozessor nicht installiert werden.

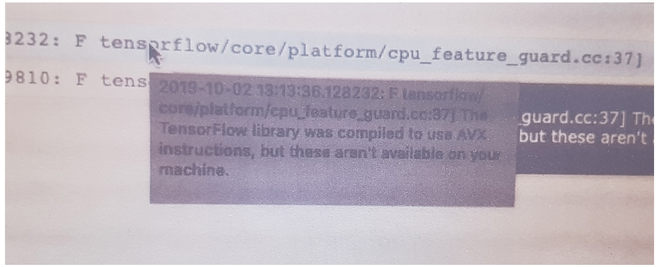


Abb. 16: Fehlende AVX Instructions Fehler

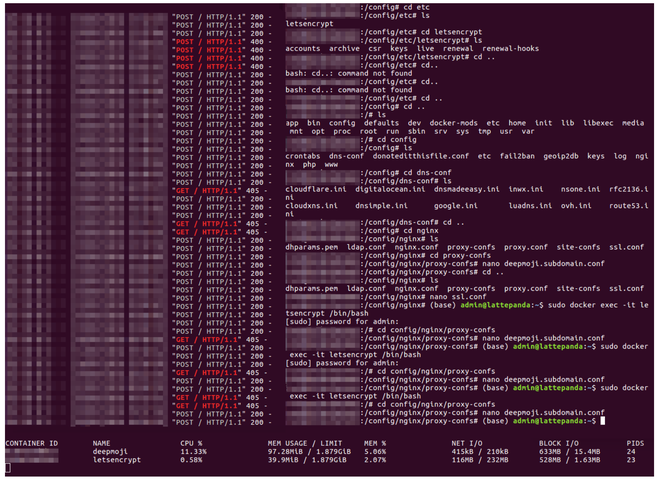


Abb. 17: DeepMoji Deployment Screenshot Collage, LattePanda Screenshot

DOCKER FILE

```
FROM python:2.7.15-stretch

RUN  apt-get update \
  && apt-get install -y \
  wget \
  unzip \
  && rm -rf /var/lib/apt/lists/*

RUN wget --no-verbose "https://github.com/bfelbo/DeepMoji/archive/master.zip" && unzip master.zip

RUN mv DeepMoji-master deepmoji
WORKDIR deepmoji

RUN pip install . && \
    pip install tensorflow pandas flask

RUN yes | python scripts/download_weights.py && mkdir api

RUN cd api && echo "" > __init__.py
COPY emoji_function.py api
COPY api_helper.py api
COPY server.py api
COPY emoji-lookup.csv .

EXPOSE 80

ENTRYPOINT [ "python", "-u", "api/server.py"]
```

ANPASSUNG - TENSOR FLOW OHNE AVX INSTRUCTIONS

```
RUN pip install . && \
    pip install --no-cache-dir https://github.com/maxhgerlach/tensorflow-1.8.0-ubuntu16.04-py27-no_avx-xeon_x5650/raw/master/tensorflow-1.8.0-cp27-cp27mu-linux_x86_64.whl pandas flask
```

AVX INSTRUCTIONS

Advanced Vector Extensions (AVX) ist eine Befehlssatzerweiterung für x86 Microprozessoren, die von Tensor Flow verwendet wird. Um Tensor Flow auf dem von mir gewählten LattePanda nutzen zu können, musste ein Weg gefunden werden, um die Abhängigkeit von dieser Befehlssatzerweiterung zu umgehen. Nach einiger Recherche konnte ich ein Git Repository finden, dass das richtige Tensor Flow wheel für die richtige Ubuntu- und Pythonversion und passende Prozessorarchitektur beinhaltet.

Für die Einbindung in den von mir benutzten DeepMoji Docker Container musste die dazugehörige Dockerfile angepasst werden. Eine Dockerfile ist im Grunde genommen eine Konstruktionsanweisung, die von Docker zum Initieren des Containers aufgerufen und befolgt wird. In dieser File finden sich alle nötigen Verweise auf Libraries und Commands die zum erflogreichen starten des Containers in seinem virtuellen Docker Enviroment notwendig sind.

Auf der linken Seite ist die ursprüngliche Dockerfile aufgeführt, die ich zum Starten des Containers benutzt habe. Darunter befinden sich die zwei Codezeilen, die ich ausgetauscht habe, um die von mir gefundene Tensor Flow Version ohne AVX Instructions zu nutzen.

Im Anschluss an diese Anpassung ließ sich der Container problemlos starten und lokal auf dem LattePanda konnten die ersten erfolgreichen Emoji-Vorhersagen berechnet werden. Nun ging es darum, diese Anfragen auch über das Heimnetzwerk machen zu können und anschließend Anfragen über das Internet zu ermöglichen.

Abb. 18: Dockerfile Anpassungen Screenshot Collage, LattePanda Screenshot

SERVER CLIENT SYSTEM

Die ersten Tests im Heimnetzwerk waren relativ schnell erfolgreich, da dieselben cURL Anfragen genutzt werden konnten, die ich bereits bei den ersten lokalen Tests auf meinem PC genutzt hatte. Hier musste einzig die lokale Netzwerkadresse ausgetauscht werden.

Um den Server vom Internet aus ansprechbar zu machen, musste jedoch noch die Portfreigabe geregelt werden, und eine eindeutige Webadresse definiert werden. Um die Portfreigabe korrekt konfigurieren zu können, habe ich mir einen neuen Router zugelegt, der die entsprechenden Einstellungsmöglichkeiten besitzt. Im Anschluss mussten diese Ports über eine konstante Webadresse erreichbar gemacht werden. Da die IP Adresse meines Netzwerks sich in gewissen Abständen ändert ist ein Dynamisches DNS (DynDNS) notwendig, um diese IP-Wechsel automatisiert und unverzüglich unter einem konstanten Hostnamen zu registrieren. Es gibt relativ viele kostenfreie Onlinedienste die DynDNS Weiterleitungen anbieten. Problematisch war nur der Umstand, dass mein Heimnetzwerk bis Dato nur unter einer IPv6 Adresse erreichbar war, welche nur unter Umständen und von manchen DynDNS Anbietern gar nicht unterstützt werden.

Diesbezüglich bin ich mit meinem Internetprovider in Kontakt getreten, um anzufragen, ob mein Anschluss auf IPv4 umgestellt werden kann. Ich konnte mich mit meiner Anfrage glücklich schätzen, da seit Ende 2019 eigentlich alle IPv4 Adressen in Europa bereits vergeben sind, die Umstellung aber dennoch bewilligt wurde und ich am nächsten Tag bereits unter der neuen Adresse erreichbar war. Die Verbindung wurde erfolgreich von mir getestet und als nächster Schritt schloss sich die Auseinandersetzung mit Datenverschlüsselung an.

SICHERHEIT

Da sich die emotionsbezogene Interpretation von nutzer*innengenerierten Texten potenziell mit sensiblen Daten auseinandersetzt und mein Konzept diesen Umstand sogar förmlich fordert, war es mir ein wichtiges Anliegen die nun via Internet versendeten Daten zu verschlüsseln. Als ersten Schritt bedeutete das, über meinen Router nur den Port 443 freizugeben, um jegliche Kommunikation mit dem nachgeschalteten Server über HTTPS laufen zu lassen. Diese Umstellung hat bewirkt, dass mein DynDNS Service künftig nur noch verschlüsselte Anfragen erfolgreich an meinen LattePanda weiterleiten kann, da keine anderen Ports zur Verfügung stehen.

Darüber hinaus, musste die serverseitige Sicherheitszertifikatsvergabe geregelt werden. Zu diesem Zweck konnte ich ein sehr nützliches Tool ausfindig machen. Let'sEncrypt stellt automatisiert Zertifikate zur Verfügung und erneuert diese auch automatisch vor ihrem Ablaufdatum. Außerdem gibt es einen offiziellen Let'sEncrypt Docker Container, der das Einrichten eines notwendigen Reverse Proxy für die Docker Umgebung vereinfacht und der ausgesprochen gut dokumentiert ist.

Um Let'sEncrypt einzurichten, habe ich den Container auf meinem Server gestartet, mich via Linux Shell in das virtuelle Environment des Containers navigiert und dort eine Subdomain Config-Datei angelegt. Diese Datei ist im rechten Screenshot zu sehen. Let'sEncrypt empfängt somit alle Anfragen, die an den Port 443 des LattePanda gerichtet sind, und leitet nur die Anfragen, die von meiner DynDNS Adresse kommen weiter an den 'DeepMoji Container. Alle anderen Anfragen werden blockiert.

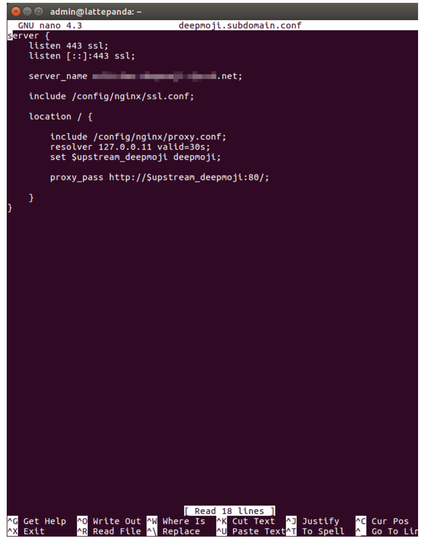


Abb. 19: Reverse Proxy Configuration File, LattePanda Screenshot

FINALE KRITERIEN

Zusammenfassend lässt sich also sagen, dass für mein Setup folgende Hard- und Softwarekonfigurationen notwendig sind:

- Einplatinencomputer mit Tensor Flow fähigem Prozessor
- CPU und Docker kompatibles Betriebssystem
- DeepMoji Docker Container
- Tensor Flow Version ohne AVX Instructions
- Router mit eingerichteter Portfreigabe
- DynDNS Adresse zur Weiterleitung der Anfragen
- Let'sEncrypt Docker Container mit eingerichtetem Reverse Proxy

Alle für diese Konfiguration benötigten Software-Elemente sind in der Abgabe enthalten. Die Konfiguration ist reproduzierbar und bietet wie eingangs erwähnt die Möglichkeit im lokalen Netzwerk oder online aufgesetzt zu sein. An diese Schritte schließt sich nun die Entwicklung einer Mobile Applikation an, die Nachrichten an diesen Server via cURL Command senden und die empfangenen Daten verarbeiten und aufbereiten kann.

UNITY NETWORKING

Unityseitig wird die HTTP Kommunikation von UnityWebRequest Objekten übernommen. DownloadHandler und UploadHandler übernehmen dabei jeweils den Daten Down- und Upload. Über die Funktion PrepareWebRequest wird ein UnityWebRequest mit einem, den an DeepMoji gerichteten Text beinhaltenden, JSON versehen. Dann wird die Serveradresse und Anfrage-Methode bestimmt. Daraufhin wird die Anfrage abgesendet und die Antwort anschließend verarbeitet.

Der auf der rechten Seite zu sehende PostHandler (Abb. 24) ist im Laufe des Projekts mehreren Refactorings unterzogen worden. Zu sehen ist hier die finale Version des Werkstücks, die weitestgehend optimiert und von redundantem Code, sowie unnötigen Abhängigkeiten befreit wurde.

Links neben dem PostHandler Code sind unstrukturierte und geordnete Versionen des Consoleoutputs in Unity zu sehen, die einmal den rohen Antwort-JSON (Abb. 22) und einmal das von meiner API verarbeitete Vorhersageobjekt zeigen (Abb. 23).

Die verbleibenden beiden Mobile-Screenshots (Abb. 20 und Abb. 21) zeigen die erste PostHandler Testversion, betriebsfähig auf dem mobilen Endgerät. Warum einige der Emojicodes von Unity nicht geparst werden können, konnte ich nicht herausfinden. Ich habe verschiedene Encoder getestet, konnte aber im besten Fall nur wenige der Emojis richtig anzeigen lassen. Da die Antwort-JSONs die Emojiklassen aber jedes Mal in derselben Reihenfolge angeben, war es möglich einfach auf ein ID-basiertes Interpretationssystem zu setzen.

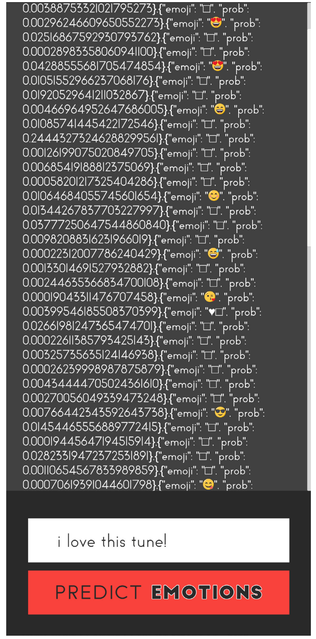


Abb. 20: Netzwerktest mit APK 01, Android

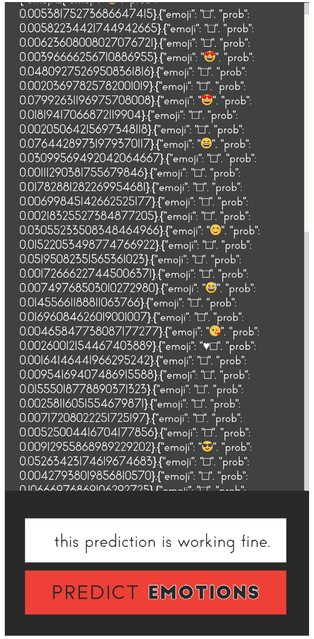


Abb. 21: Netzwerktest mit APK 02, Android

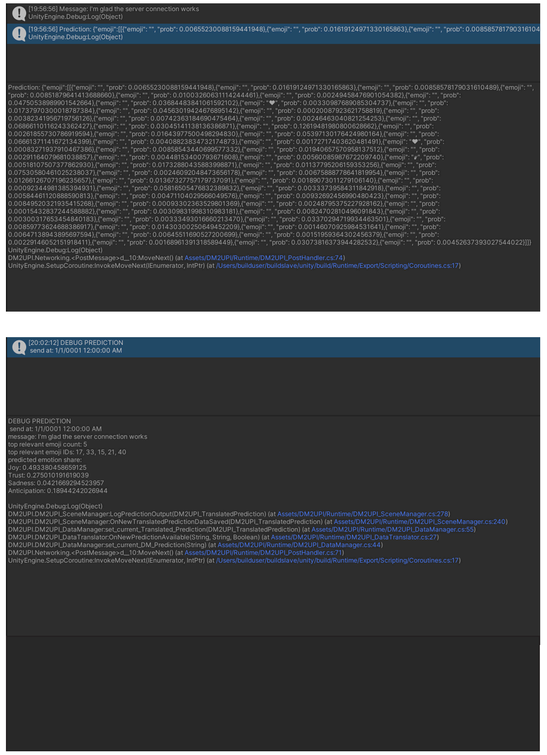


Abb. 22: Netzwerktest in Unity 01, Unity Screenshot

Abb. 23: Netzwerktest in Unity 02, Unity Screenshot

Abb. 24: Eigener Netzwerk-Code, VisualStudio Screenshot

GESELLSCHAFTLICHE RELEVANZ

Wie eingangs erwähnt, ist die in diesem Designprojekt behandelte Grundthematik ein mit Vorsicht zu behandelndes Thema. Technologische Neuerungen, die große Datenmengen zum Training komplexer AI-gorithmen benutzen und in das zwischenmenschliche Kommunikationsverhältnis eingreifen setzen besondere Sorgfalt bei der Entwicklung voraus und sollten moralischen Richtlinien unterliegen. Die Arbeit an dem Themenfeld des maschinengestützten Emotional-Understandings ist dabei keine Ausnahme. Dementsprechend möchte ich die Ergebnisse meiner Prototypen im Rahmen der angewandten Medienkompetenzentwicklung verorten, indem ich mich darum bemühe die Funktionalität des benutzten Modells unmittelbar erfahrbar zu machen und konfrontativ in den Mittelpunkt der Interaktion zu stellen. Die API als Hauptwerkstück hingegen soll an dieser Stelle mit dem Appell an nachfolgende Entwickler*innen versehen sein, ähnliche — wenn nicht ausführlichere — Kriterien für die Zweckbindung des hier für die Unity Entwicklung zugänglich gemachten Deep Learning Modells anzuwenden.

Ein konstruktiver Gedankenspaziergang, den man als Designer oder Entwickler diesbezüglich unternehmen kann, ist, sich über das wesentliche Problemlösungs-Mindset unseres Arbeitsfeldes bewusst zu werden. Nolen Gertz führt in seinem Buch Nihilism and Technology an, dass Effizienz und Objektivität Eigenschaften sind, die wir in technologischen Lösungsansetzen wertschätzen, dieser Bewertungsrahmen aber gleichermaßen dazu führt, dass wir Menschen als ineffizient und voreingenommen wahrnehmen und somit als Problem sehen, dass es durch Technologie zu lösen gilt (vgl. Gertz 2018, S. 3).

> "[...] technologies both shape and are the result of [...] ideological definitions." (Gertz 2018, S. 7)

Technologien sind ohne Kontext keine moralischen Entitäten, lassen sich aber auch nicht ohne Zweckgebundenheit betrachten. Bei der Entwicklung neuer Technologien darf das "wofür" also nicht auf der Strecke bleiben und um das zu gewährleisten, müssen wir uns, wie Gertz anführt, darüber unterhalten, wie wir Begriffe wie "Fortschritt", "besser" oder "schlechter" in diesem Zusammenhang definieren (vgl. Gertz 2018, S. 7). Das Aufkommen von Social Media hat weltweit Millionen von Menschen miteinander vernetzt, hochmoderne AI-gorithmen entscheiden darüber welche Nachrichten, Mitteilungen, oder Werbung jede*r einzelne Nutzer*in tagtäglich zu Gesicht bekommt, aber kann dieser kommunikative Wandel, der Informationsblasen und selbst bestätigende, mediale Echokammern sondergleichen hervorgebracht hat aus gesellschaftstheoretischer Sicht als Fortschritt bezeichnet werden? Algorithmisierte Datingapps preisen an, anhand der Nutzer*innen-Merkmale perfekt kompatible Partner*innen aus dem Mitglieder*innen-Pool zu ermitteln, aber welche romantisierte Theorie menschlicher Beziehungen steht hinter der Annahme, dass merkmalsgleiche Personen besser zueinander passen, oder es gar den einen perfekt kompatiblen Menschen gibt?

Während es nahe liegt Technologien per se als Lebensqualität verbessernde Werkzeuge zu sehen, ist es, besonders wenn man an der Entwicklung beteiligt ist, wichtig, von Zeit zu Zeit die Perspektive zu wechseln und wahrzunehmen wie Technologien gelegentlich überholte Weltbilder verfestigen können.

Hinsichtlich Deep Leerning beschäftigen sich zunehmend Institute und Gruppierungen wie das AINOW Institut, das Alan Turing Institut und das NoBIAS Projekt mit der Untersuchung systeminhärenter menschlicher Vorurteile innerhalb künstlicher Intelligenz, sowie der Auswirkung und Prävention vorurteilsbehafteter Algorithmen. Die Arbeitsergebnisse dieser Initiativen zeigen einmal mehr, wie wichtig es ist die systemischen Auswirkungen der eigenen Arbeit frühzeitig zu betrachten, transparent zu Arbeiten und Entstehungskontexte eigener Produkte kritisch zu reflektieren.

EMOTIONEN MODELL

An die, durch DeepMoji vorhergesagten, Emojis sollte sich für die breite Nutzbarmachung der Ergebnisse eine Emotionen-Kategorisierung anschließen. Die API, die als Werkstück entstanden ist, stellt zu diesem Zweck unvoreingenommene Konfigurationsoptionen zur Verfügung. Um für das Prototyping jedoch eine Ausgangsposition zu haben, habe ich nach vorhandenen Modellen gesucht, die sich strukturell eignen, um mögliche Spielabläufe und Visualisierungen auf ihren Grundannahmen fußen zu lassen.

Das rechts zu sehende Wheel of Emotions nach Robert Plutchik stellt diesbezüglich einen vielversprechenden Ausgangspunkt dar. Die Anordnung der Emotionen in Pluchiks Modell gleichen einem Farbkreis, indem konträre Emotionen wie Komplementärfarben gegenüberliegend platziert sind. Jede Emotion ist zusätzlich in unterschiedlichen Intensitäten aufgeführt und in der Weise sortiert, dass weiter außen stehende Emotionen schwerer unterscheidbar sind, als weiter innen stehende Emotionen. Ähnlich wie Farben eines Farbkreises lassen sich nach Plutchik auch Emotionen miteinander mischen, um neue Emotionen zu formen. So entsteht Liebe beispielsweise aus Vertrauen und Freude, Reue hingegen aus Abscheu und Traurigkeit. (vgl. o.V. Wikipedia 2020)

Die in Plutchiks Modell beschriebenen Regeln für die Positionierung, Ableitung und Verhältnismäßigkeit seiner Primäremotionen und der daraus geformten primären und sekundären Dyaden zweiteiliger kombinierter Emotionen legt ein solides Fundament für Parametrisierung und Verrechnung der im Anschluss an eine DeepMoji Vorhersage zugewiesenen Emotionsprozentwerte.

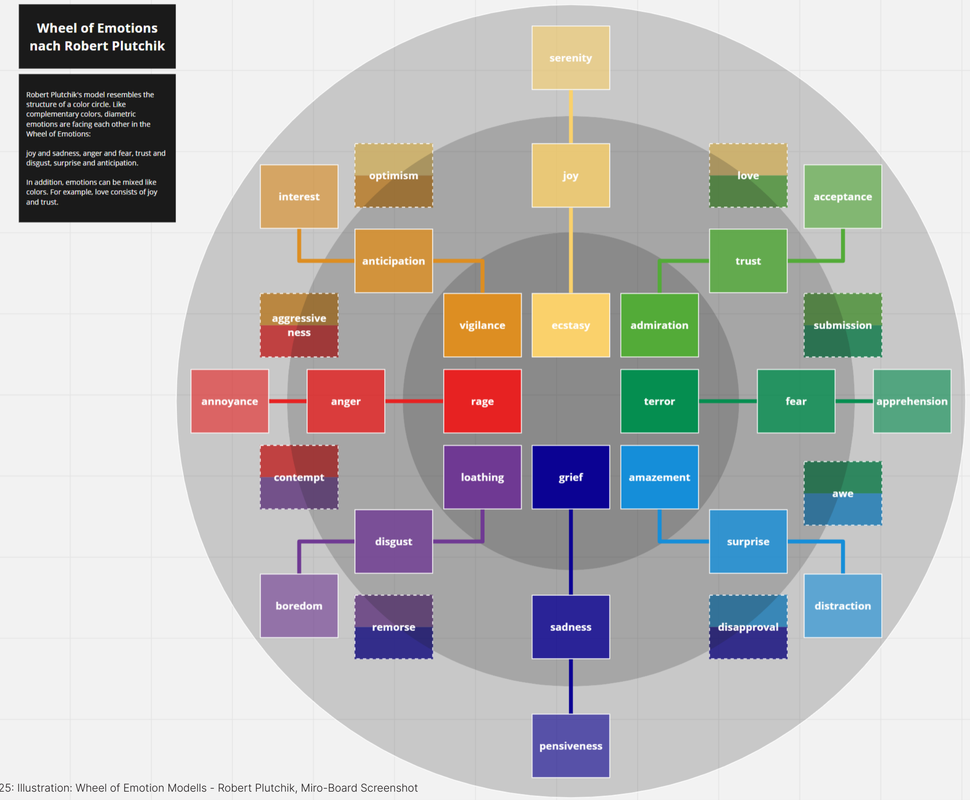


Abb. 25: Illustration: Wheel of Emotion Modells - Robert Plutchik, Miro-Board Screenshot

UMFRAGE UND BALANCING

Um das Balancing der Emotionen nicht ausschließlich auf einem Bauchgefühl beruhen zu lassen und somit einen wichtigen Schritt in der Nutzung dieses auf Emojis trainierten Modells zu übergehen, wurde während der Projektzeit ein Pretest durchgeführt. Die Umfrage umfasste 64 Fragen, in welchen zu jedem in DeepMoji verankerten Emoji eine beliebige Anzahl von 32 Emotionen zugeordnet werden konnte. Die Emotionen wurden farblich kategorisiert, um die Antwortmöglichkeiten übersichtlicher zu gestalten. Trotz dessen blieb die Umfrage sehr umfangreich und wurde im Durchschnitt in knapp 44 Minuten von den Befragten abgeschlossen. Die Umfrage ist über den nebenstehenden QR-Code einsehbar.

Die breite Auslegung des Emotionsrasters brachte anschließend auch die Notwendigkeit mit sich, die einfache Zuordnung einer Emotion pro Emoji zu erweitern, mehrere Emotionen zuzulassen, eine Gewichtung unter den zugeordneten Emotionen zu erlauben und eine Codelösung zu finden, die es ermöglicht dynamisch neue Emotionen zu erstellen, ohne die zuvor erstellten Enums im Code selber anpassen zu müssen. All diese Erkenntnisse um benötigte Funktionalitäten wurden besonders bei der Entwicklung meiner API zwingend erforderlich, um Entwickler*innen ein kräftiges Tool an die Hand zu geben, das sofort einsatzbereit ist. Die Umfrage selbst zu erstellen brachte einige Rechercheaufgaben mit sich, zu denen unter anderem gehörte verschiedene Onlineumfrageanbieter zu evaluieren und alternative Formen der Erhebung, wie zum Beispiel die Nutzung einer Imagemap, zu testen. Da sich der Ideation- und Prototypingprozess parallel weiterbewegte und sich das Werkstück von einer konkreten Applikation hin zu einer API entwickelte, mussten auch an dieser Stelle Prioritäten gesetzt werden. Die Empfehlung eine, auf die spezifische Zielgruppe einer, diese API benutzenden, Applikation ausgerichteten, Umfrage durchzuführen, wurde in die Dokumentation der API mit aufgenommen.

Durch mein erstes Bachelorstudium der Kommunikationswissenschaft ist mir die Dimension einer repräsentativen Umfrage zu diesem Themenkomplex bewusst. Um valide Ergebnisse zu erhalten, den Einfluss von Störvariablen zu minimieren und die demografische Repräsentativität zu gewährleisten wäre eine ausgedehnte Studie notwendig, die in sich selbst eine eigene Bachelorarbeit darstellen könnte. Aus diesem Grund habe ich die Erhebung im Anschluss an den Pretest eingestellt, jedoch wie oben aufgeführt zahlreiche wichtige Anstöße für den Featurekatalog meiner API mitgenommen.

Abb. 26: QR Code Link zu Umfrage

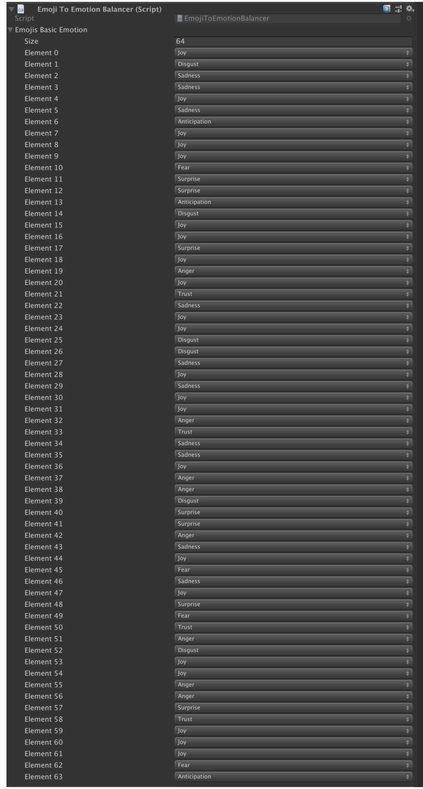


Abb. 27: Erste Emotionen zu Emojis Zuordnung, Unity Screenshot

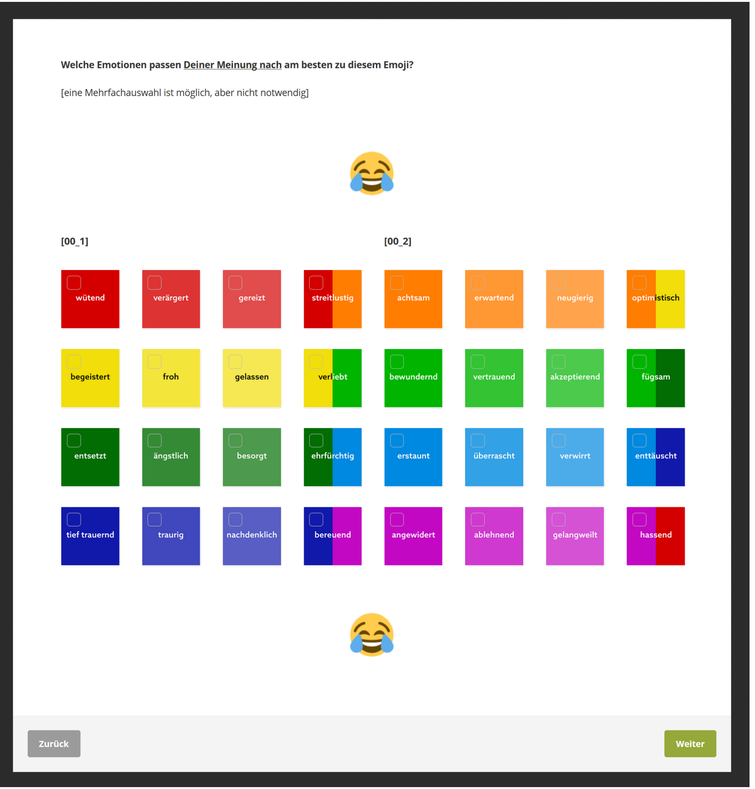
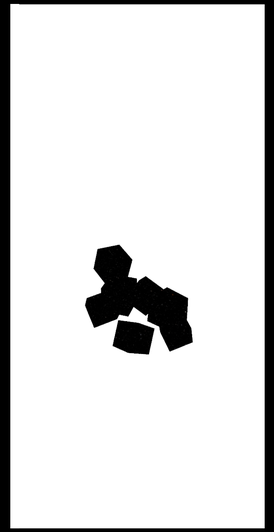


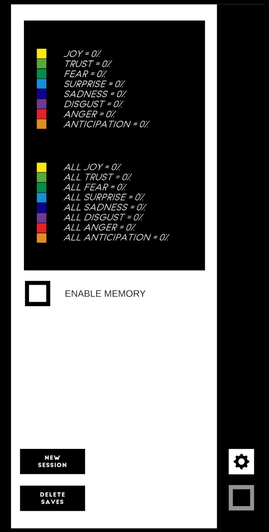
Abb. 28: Beispielfrage: Emotionen zu Emojis Umfrage, Browser Screenshot

PROTOTYPING

Der Prototyping Prozess war von vielen Iterationsschritten geprägt, die von unterschiedlichen Zweckbindungen durch verschiedene Visualisierungsformen hin zu der Konzentration auf die eigentliche Code-Schnittstelle führten und somit in der Entwicklung des Haupt-Werkstücks, des DeepMoji to Unity Programming Interface's mündeten.

Die Untersuchung, Reflexion und Revision der unterschiedlichen Ansätze war damit zentraler Bestandteil des Arbeitsprozesses, was zwar stets eine Aktualisierung der Zielsetzung mit sich brachte, das Ergebnis aber umso gefestigter werden ließ.





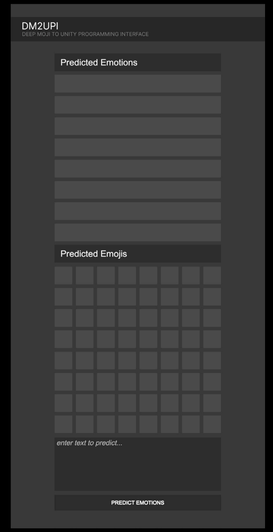


Abb. 29: UI Shader digital Prototyping, Unity Screenshot

Abb. 30: Farben Prototyp Debugging Menü, Unity Screenshot

Abb. 31: DM2UPI Demo Szene, Unity Screenshot

ÜBERBLICK Das Prototyping der Visualisierungen und der dynamischen Interpretation der Vorhersageergebnisse war in langen Stücken eng mit der unityseitigen Weiterentwicklung der Interpretationslogik verbunden. Wie bereits erwähnt, reicht die einfache Adressierung des DeepMoji Servers und die Weitergabe der empfangenen JSON Daten nicht aus, um alle relevanten Informationen für einen interaktiven Showcase maschinell-emotionalen Verstehens zu entwerfen. Somit begann die Prototypingphase dieses Projekts mit dem Entwurf von exemplarischen Code-Strukturen und Klassendiagrammen.

Zu einem soliden Prototypen gehören zudem aber auch ein durchdachtes User Interface, ein ansprechendes und nachvollziehbares Veranschaulichungskonzept sowie umfangreiche technische Überprüfungen und Playtests.

Zu den Arbeitspaketen, die mit jedem neuen Ansatz einhergingen, gehörten also:

- **Anpassung der Vorhersage-Interpretationslogik**
- **UI und UX Entwürfe und Anpassungen**
- **Revision der Zielplattform und Zielgruppe**
- **Abwägung des inhaltlichen Gehalts mit Bezug auf das Themenfeld der Mensch-Maschine Kommunikation**

Im Entwicklungsprozess stellte sich heraus, dass sich das Designprojekt in diesem Umfang entweder zeitnah auf einen konkreten Ansatz oder auf einen relevanten und nachhaltigen Teilbereich konzentrieren sollte, um überzeugende Ergebnisse zu produzieren.

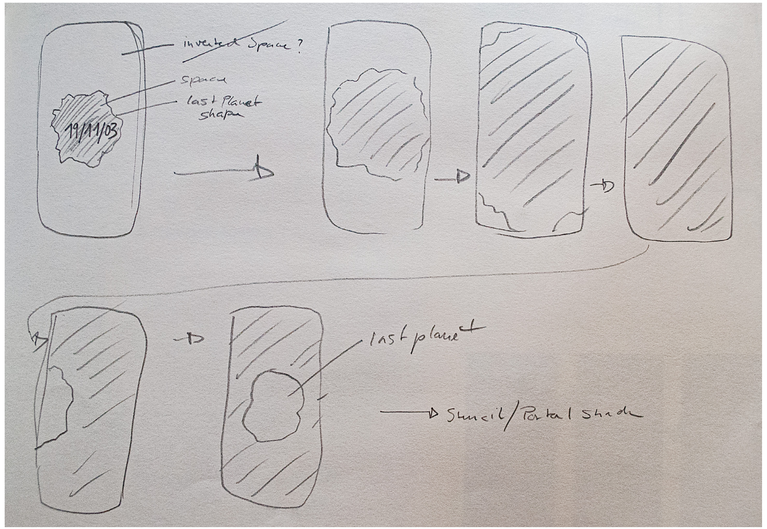


Abb. 32: UI/UX Skizze Lernjournal

TAGEBUCH Das erste Anwendungskonzept, das auch schon im Exposé umrissen wurde, stellt die Interaktionsstruktur eines virtuellen Tagebuchs in den Mittelpunkt. Dabei sollen die Nutzer*innen die Möglichkeit haben, die Applikation jeden Tag mit kurzen Nachrichten zu versorgen, die gesammelt interpretiert werden. Jeder Tag hat dabei eine separate Visualisierung der, in den Nutzer*innentexten erkannten, Emotionen und spiegelt somit durch einen konzentrierten positiven Feedbackloop das maschinelle Verständnis der Inputs wider.

Verstrichene Tage lassen sich rückschauend betrachten, aber nicht mehr verändern, um — einem Tagebuch gleich — klare zeitliche Zuordnungen der eingegebenen Texte zu erlauben und die Konditionierung des regelmäßigen Eintragens neuer Texte zu fördern.

Der rechts stehende Entwurf zeichnet sich durch schlichte Form- und Farbgebung aus, um sowohl Form als auch Farbe als sinntragende Elemente in den Emotionsdarstellungen zu verorten und Doppeldeutigkeiten im Design zu vermeiden.

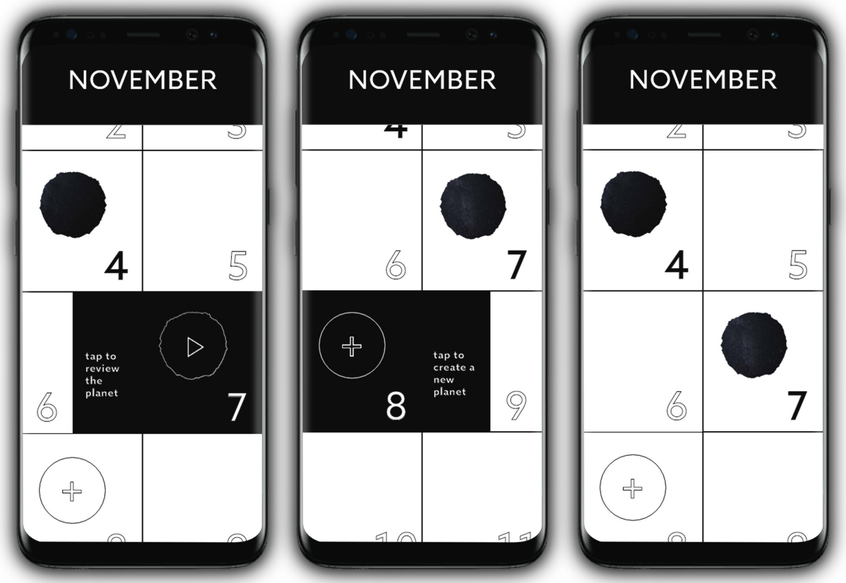


Abb. 33: Tagebuch Hauptmenü und Kalenderansicht UI Mockups

Die Intimität eines Tagebuchs mit der algorithmisierten Interpretation der Maschine zu verbinden, stellt dabei eine interessante Gegenüberstellung zweier die Privatsphäre bedingenden Bereiche zeitgemäßer Mediennutzung dar. Um die Annäherung an die Zusammenführung dieser Bereiche zugänglicher zu gestalten, ist bewusst das Smartphone als Plattform ausgewählt worden. Kein anderes Device wird von seinen Nutzer*innen auf die gleiche taktile Art und Weise bedient, ist stetiger Begleiter durch den Alltag und wird zu Eingabezwecken förmlich gestreichelt. Die vertraute Beziehung zu diesem Gerät kann also helfen Berührungsängste mit neuen Technologien zu verringern, birgt aber auch in erhöhtem Maße die Gefahr des Missbrauchs durch Datenveruntreuung, wie er seit einigen Jahren in vielen Applikationen nachgewiesen wurde.

Um mit dem Projekt SELF-INFLICTED EMPATHY keine unbedachten Schritte zu gehen, soll der emotionale Ausdruck der Applikation vorerst auf die direkte Interpretation der eingegebenen Texte gerichtet sein. Ich sehe darin für viele Nutzer*innen die Möglichkeit, erste Schritte des erweiterten Verständnisses der Potentiale unserer Geräte zu machen, während die meisten Anwendungsfelder von Emotionsinterpretationsalgorithmen hinter verschlossenen Türen entwickelt werden. Medien- und Technologiekompetenzen lassen sich in den seltensten Fällen ohne Praxisbezug entwickeln.

Ein sich daran anschließender Gedanke in der Ideationphase war, RESTful APIs gängiger Social Media Netzwerke einzubinden, um den Nutzer*innen zu ermöglichen, ihre Posts auf Twitter, Facebook oder Instagram direkt in ihr Tagebuch einfließen zu lassen - eine Option, die für zukünftige Projekte immer noch interessant ist, durch die Fokussierung auf andere relevante Teilaspekte aber an dieser Stelle nicht weiter entwickelt wurde.

Auf der rechten Seite sind Mockups der Übergangsanimation vom Auswahlmenü eines bestimmten Tages des Tagebuchs hin zum Gameview aufgeführt. Dort angekommen war ein übersichtliches Userinterface mit einer einfachen Eingabezeile geplant, in die ungefähr Tweet-lange Texte eingefügt werden können. Das Abschicken des Textes wirkt sich nach kurzer Zeit auf die Visualisierung aus. Wird das Telefon horizontal gehalten lassen sich alle für diesen Tag eingegeben Texte erneut lesen. Die Veranschaulichung der Emotionen durch ein dreidimensionales Objekt zu demonstrieren, hat zudem die Überlegung mit sich gebracht, AR Technologie zu nutzen, um das Objekt durch das Tiefenverständnis des Geräts im Raum zu platzieren, ohne dabei den, bei AR Erfahrungen überwiegend genutzten, Kamerahintergrund zu verwenden. Die, so in der analogen Welt verankerte, räumliche Position als physisch erlebbaren Abstand oder gewählte Nähe erfahrbar zu machen war eine weitere Idee, die technischen Besonderheiten der gewählten Plattform zu nutzen, um die Intensität der Erfahrung zu verstärken.

Da sich die Zielsetzung des Projekts im Projektzeitraum mehrfach verändert hat, sind einige dieser Features jedoch nur Teil des Konzepts. Der spätere Fokus auf die Entwicklung einer umfangreichen API hat die Funktion der entwickelten Prototypen auf erweiterte Proof of Concepts und Veranschaulichungsapplikationen reduziert.

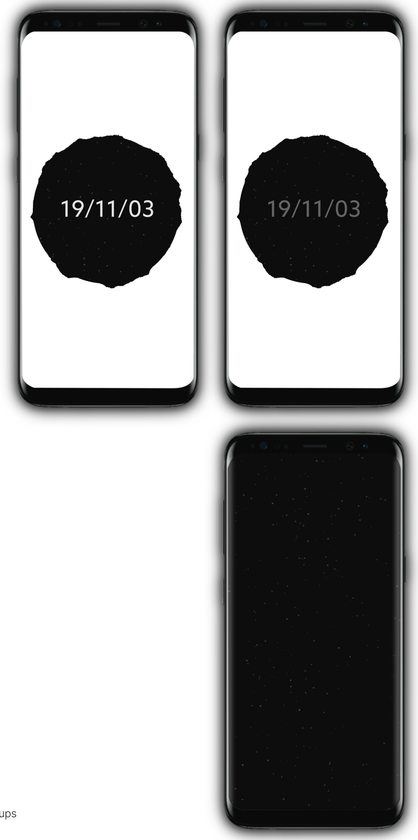


Abb. 34: Tagebuch Hauptmenü zu Gameview Animation Mockups

PLANETEN Der erste Visualisierungsansatz war den Text Input in einem täglichen, kleinen Planten mit variierender Topografie zu manifestieren. Es handelt sich dabei um ein formbares Objekt, das mit Lebendigkeit assoziiert wird und reichlich Merkmale birgt, die sich parametrisieren lassen. Distinkte Planetenformen und Farben werden beispielsweise im Film oft genutzt, um Charakteristiken und Mentalität der bewohnenden Lebensformen abstrahiert zu illustrieren. So ließe sich jede Woche ein überschaubares Universum von sieben Planeten formen, durch das die Nutzer*innen retrospektiv navigieren können.

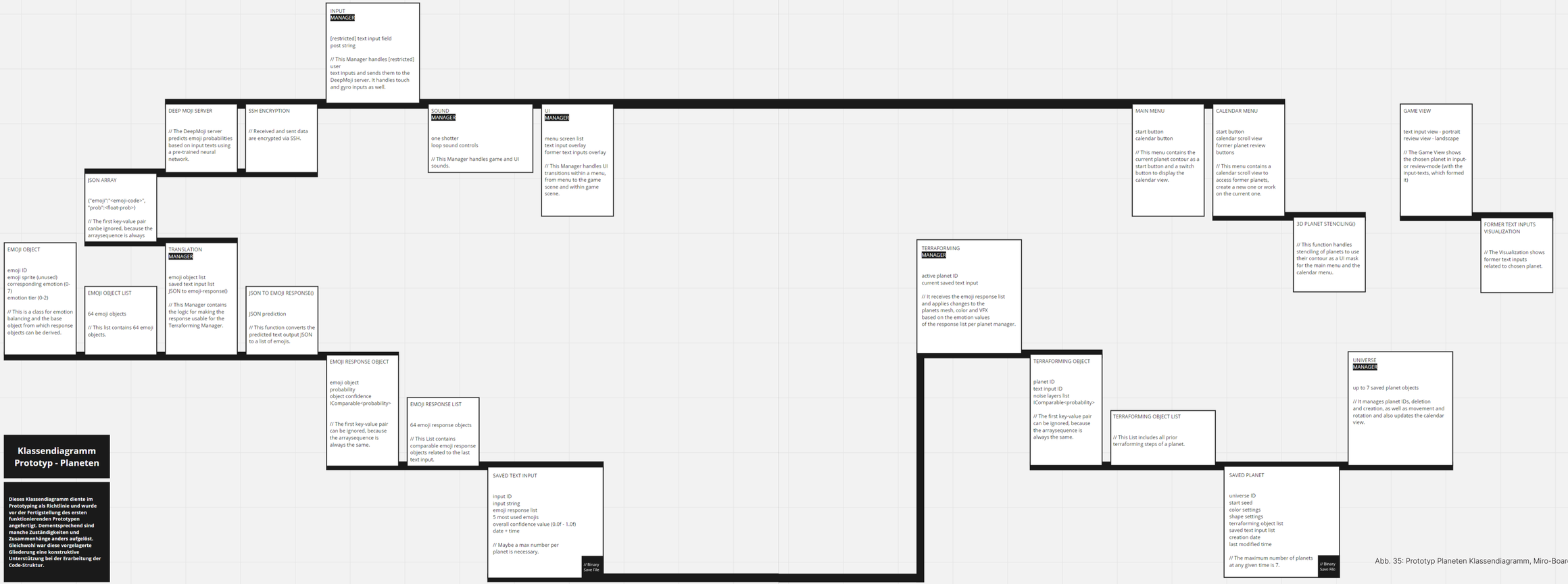
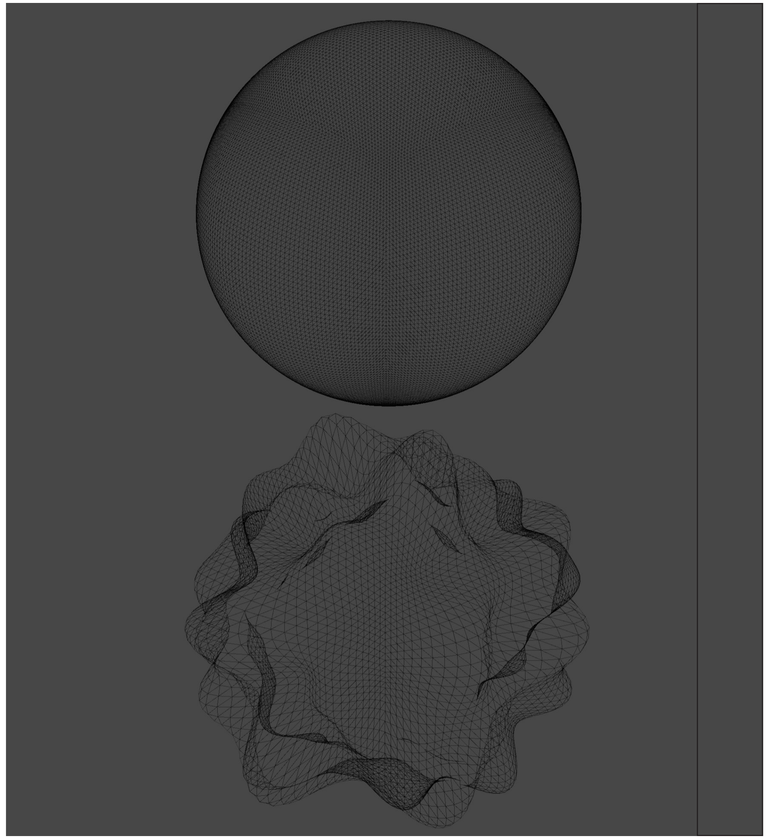


Abb. 35: Prototyp Planeten Klassendiagramm, Miro-Board Screenshot



Die ersten Ideen, welche Darstellungsmöglichkeiten der eigegebenen Emotionen sich durch das Abbild eines Planeten nutzbar machen lassen würden, umfassten unter anderem:

- die allgemeine Form, mit Erhebungen und Vertiefungen, sowie die Verhältnismäßigkeit von Land- und Wasserbereichen
- die Farblichkeit jener Bereiche und Größe in verschiedenen Klimazonen
- dynamische Effekte in Wetter und Tektonik
- Pflanzenwachstum und bewegliche Agenten als Lebewesen mit bestimmten Verhalten

Um mich nicht in der Fülle an Möglichkeiten zu verlieren, habe ich zuerst darauf konzentriert ein System für die prozedurale Generierung von Planeten zu erstellen, dass die minimal notwendigen Visualisierungsparameter enthält und sich auf Form- und Farbgebung fokussiert. Nach einiger Recherche konnte ich ein sehr hilfreiches Tutorial finden, das sich genau mit dieser Thematik beschäftigt. Diese Videoreihe des YouTubers Sebastian Lague ist unter dem Kapitel “Verwendete Hilfsmittel” dieser Arbeit aufgeführt und bildete den Ausgangspunkt für den Code, der für die hier zu sehenden Planeten genutzt wurde.

Das auf Co-Coding angelegte Tutorial behandelte zuerst die aus einer Würfelform durch Vertexmanipulation erstellte Kugel mit variabler Auflösungseinstellung und das adäquate Rendering, welches die, durch die grundlegende Würfelform entstehenden Seems verdeckt. Anschließend wurden mehrere Formen unterschiedlicher Noise Verformungen genutzt, die auch als Filter füreinander verwendet werden können, um kontinentartige Formen auf der Kugel entstehen zu lassen. Als nächster Schritt wurden Farb- und Formsettings als Scriptable Objects eingeführt, die, ergänzt durch IMGUI Editor Scripting, variable Konfigurationen der Planetenoberfläche ermöglichen. Abschließend wurde die Erstellung eines Shaders via Unity ShaderGraph behandelt, der die Faces des Meshs anhand der Höhenlage, Position in einem definierten Biom, oder Entfernung von der Küstenlinie anhand mehrerer Gradienten einfärbt.

Abb. 36: Prototyp Planeten Formengenerierung, Unity Screenshot

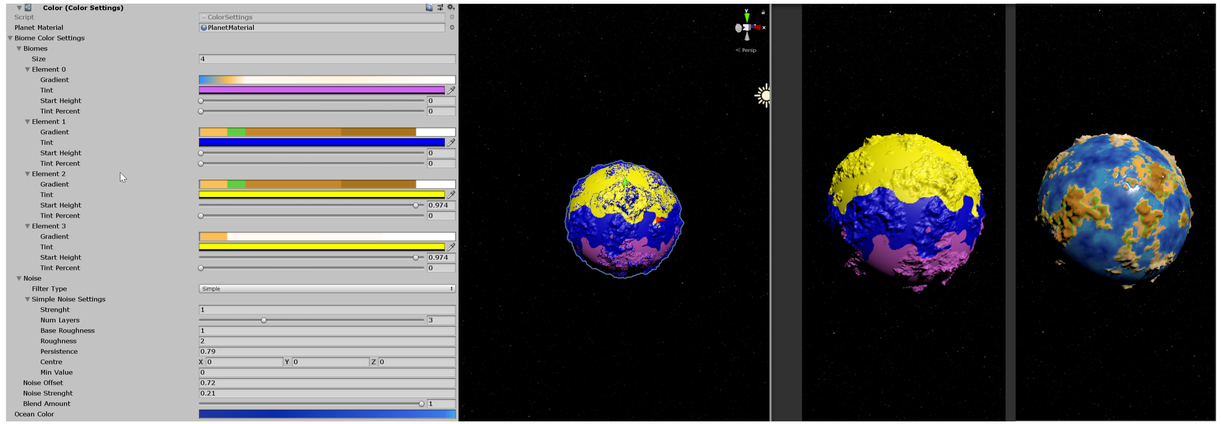


Abb. 37: Prototyp Planeten Klimazonengenerierung, Unity Screenshot

Die am Factory Method Pattern orientierte Herangehensweise der Planetengenerierung war ein lehrreicher Exkurs, der mir viele Instrumente für die prozedurale Generierung einzigartiger Formen aufgezeigt hat. Es galt allerdings weiterhin zu bedenken, dass eine Fülle an Darstellungsoptionen nicht unbedingt dienlich für den hier umsetzbaren Projektumfang sein muss. Oben stehend ist ein Screenshot der Color Settings zu sehen, die, nach dem Filtern durch drei verschiedene Klimazonen, die Färbung der Planetentopografie bestimmen. Ähnlich wie auch bei den Shape Settings sind an dieser Stelle viele manuelle Voreinstellungen nötig, um einen konsistent aussehenden Planeten zu generieren, der als solcher erkannt werden kann. Alle diese Voreinstellungen müssen durch händische Tests in Spielräume und Wertebereiche mit Mindest- und Maximalausprägungen eingeteilt werden, um eine automatisierte Generierung durch die emotionsgebundenen Vorhersagewerte nutzer*innengenerierter Texte zu ermöglichen. Dabei müssen auch die Überschneidungen von Extremwerten bedacht werden, die aus der Kombination aller potenziellen Einstellungsoptionen hervorgehen können. Für diese Wertebereiche müsste es auch klare Kategorisierungen geben, die eindeutig bestimmten Emotionen oder Stimmungen zugeordnet werden können und als solche identifizierbar bleiben.

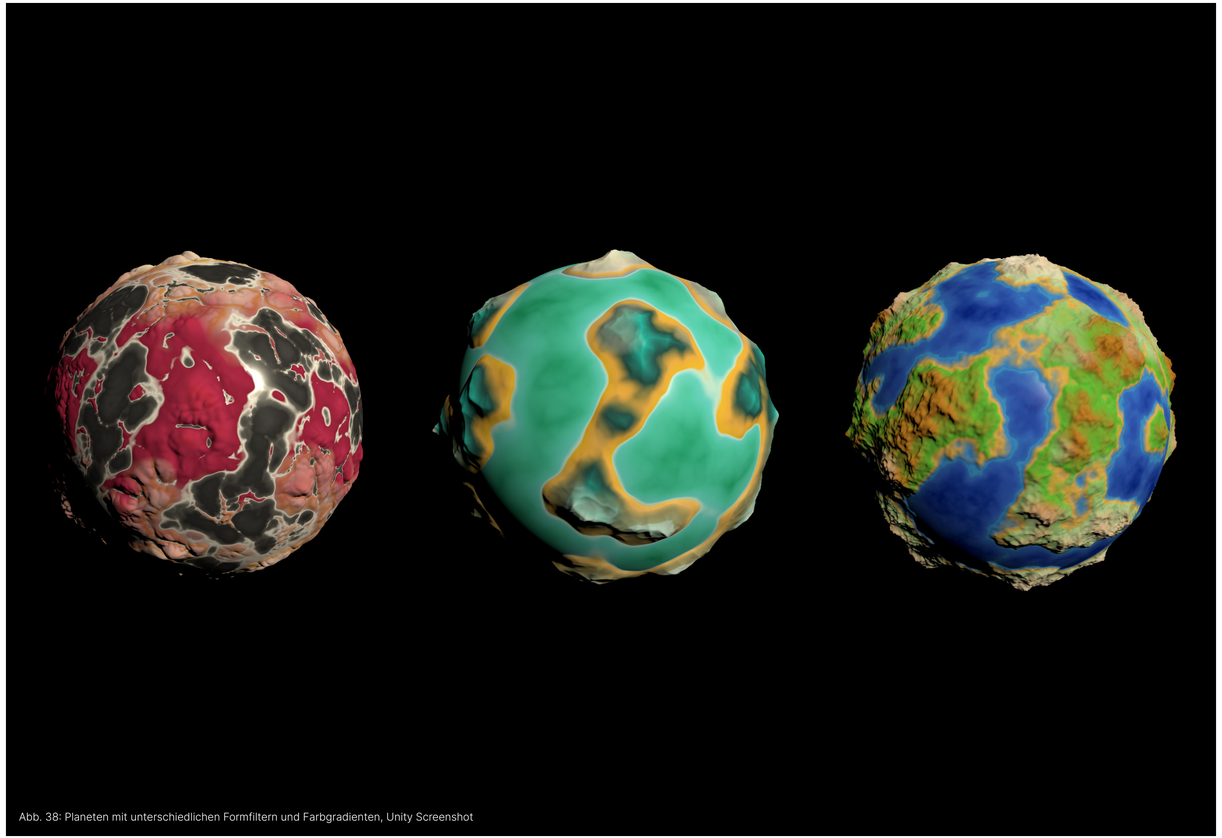
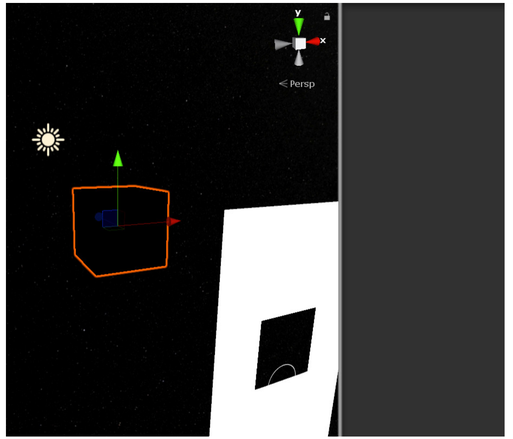
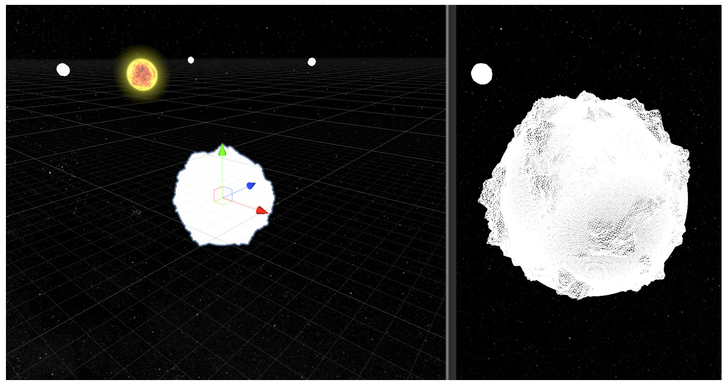


Abb. 38: Planeten mit unterschiedlichen Formfiltern und Farbgradienten, Unity Screenshot





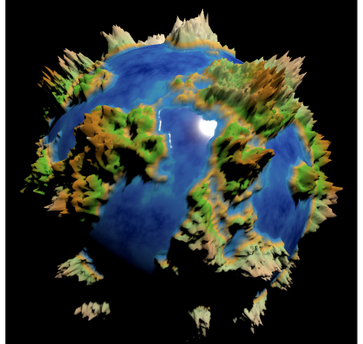
Um den Ansatz der UI Mockups umzusetzen und einen passenden Übergang zwischen dem schlichten Haupt- und Kalendermenü zum visuell komplexeren Gameview zu erstellen, habe ich mich mit der Einbindung eines Masking Shaders befasst, der die Umrisse der nutzer*innengenerierten Planeten als Schablone verwendet, um Löcher in das UI Interface zu stanzen, durch die die dahinter liegende Szene, eine sternenbedeckte Sky-Box, zu sehen ist.

Durch die Auswahl einer solchen Silhouette im Hauptmenü sollte sich diese vergrößern, bis die gesamte Szene sichtbar wird. Anschließen würde sich ein Kameraschwenk zum ausgewählten Planeten, zu dem der Umriss gehört. In der Szene sollten darüber hinaus in angemessener Entfernung alle anderen Planeten der laufenden Kalenderwoche zu sehen sein. Die anderen Planeten sind somit auf eine Anzahl von maximal sieben beschränkt und sollten als Szenenelemente weniger Aufmerksamkeit auf sich ziehen, als viel mehr die Kohärenz der Anwendungswelt unterstreichen. An dieser Stelle habe ich auch überlegt, wie mit dem zentralen Stern als Ankerpunkt und Lichtquelle jedes wöchentlichen Universums umzugehen ist. Wenn kein konkreter Bezug, zum Beispiel über die Visualisierung gesammelter Emotionswerte aller Planeten, zum Interaktionsflow und Anwendungsloop herzustellen ist, sollte auf solche naturalistischen Elemente aber besser verzichtet werden, um eine klare visuelle Kommunikation der Applikationsfunktionalitäten und ein störungsfreies Aufmerksamkeitsmanagement zu gewährleisten.

Der Fokus könnte an dieser Stelle eher darauf liegen die kontemplativen Elemente der Anwendung in den Mittelpunkt zu stellen, indem der ausgewählte Planet durch Kamerabewegungen und Zoomstufen erkundbar wird.

Abb. 39: Stencil Shader Tests 01, Unity Screenshot

Abb. 40: Planet in Universum, Unity Screenshot



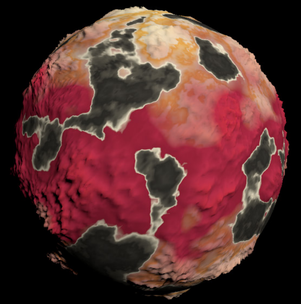


Abb. 41: Planeten mit farbigem Material, Unity Screenshot

LESBARKEIT

Wie aus den letzten Seiten hervorgeht, haben sich mit dem Prototyping des planetaren Visualisierungsansatzes zahlreiche Entwicklungsbaustellen aufgetan, deren Priorisierung für die Weiterentwicklung erforderlich wurde.

In einem betreffenden Beratungsgespräch mit meinem betreuenden Professor wurde ich darauf hingewiesen, dass die generierten Planetenformen auf Nutzer*innen beliebig wirken könnten und sich der Detailreichtum negativ auf die Lesbarkeit der Emotionsvorhersagen auswirken kann.

Als Simplifikation können Formen und Farben durchaus lesbar bleiben, allerdings bringt die Verdichtung auf eine Planetenform auch Implikationen mit sich, die der Eindeutigkeit der Vorhersagen schaden, den technischen Prozess intransparenter machen und von der direkten Interaktionserfahrung zwischen Mensch und Maschine abtragen, anstatt diese zu bereichern.

Ob die bereits erwähnten Konfigurationscluster aus Minimal- und Maximalwerten, Farbgradienten und topologischer Verformung überhaupt decodierbare Ergebnisse liefern würde, war unklar und angesichts der Menge an potenziellen Kombinationsmöglichkeiten im Rahmen des Designprojekts auch schwer test- und ermittelbar.

An dieser Stelle entschloss ich mich also die Arbeit an dem Plantenprototypen einzustellen und mich auf elementare Visualisierungsmöglichkeiten in Form und Farbe zu konzentrieren, die die eindrucksvollen Fähigkeiten des zugrunde liegenden Deep Learning Modells erkennbarer und spielerischer erfahrbar machen können.

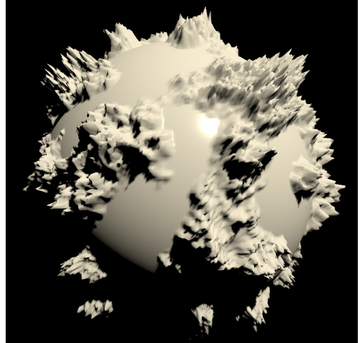
FORMEN

Die Formengenerierung der Planten hat einen möglichen Ansatz der Simplifizierung geboten, den ich in einzelnen Visualisierungstests evaluiert habe. Dazu gehörte die Auflösung des generierten Meshes inkrementell zu erhöhen oder zu verringern und minimalistische Shader zu nutzen.

Die Auflösung der Meshes lässt sich von einem einfachen sechsseitigen Kubus mit zwei Tris pro Face bis zu 1.700 Tris skalieren. Eine solche hohe Auflösung ist allerdings gerade für mobile Plattformen sehr unperformant. Im niedrigschwelligen Bereich der Auflösung ließen sich jedoch beispielsweise mit einfachen Wireframe Shadern interessante visuelle Ergebnisse erzielen, die auf den folgenden Seiten zu sehen sind.

Im Endeffekt habe ich diesen Ansatz aber nicht weiter verfolgt, da die Parametrisierung sich immer noch an der vorgegebenen Planetenform orientieren musste und die Emotionsvorhersagen sich somit effektiv nur auf den Detailgrad der Form auswirken konnten. Höhenunterschiede wie Erhebungen oder der ursprüngliche Meeresspiegel wurden weiterhin durch komplexe, schwer emotional-kategorisierbare Einstellungen erzeugt und die Ausgangsform war und blieb ein Planet.

Darüber hinaus stellte sich zudem die Aufgabe eine adäquate Formensprache zu finden, die sich, entsprechend dem Feedback zur Lesbarkeit, durch intuitives Verständnis und klare Unterscheidbarkeit der sie beeinflussenden Emotionen auszeichnet. So wirkte die Hinwendung zu einer auf reiner Farblichkeit basierenden Veranschaulichung für das Designprojekt ertragreicher.



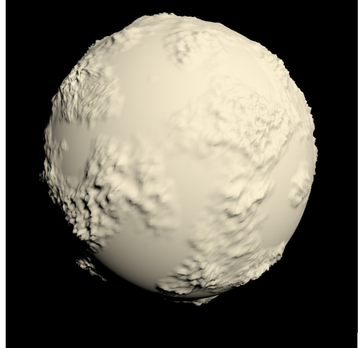
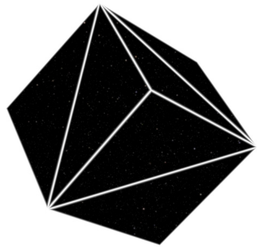


Abb. 42: Planeten ohne farbiges Material, Unity Screenshot



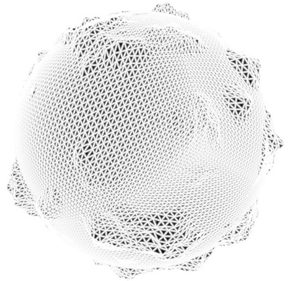


Abb. 43: Planeten verschiedene Auflösungsstufen Collage 01, Unity Screenshots

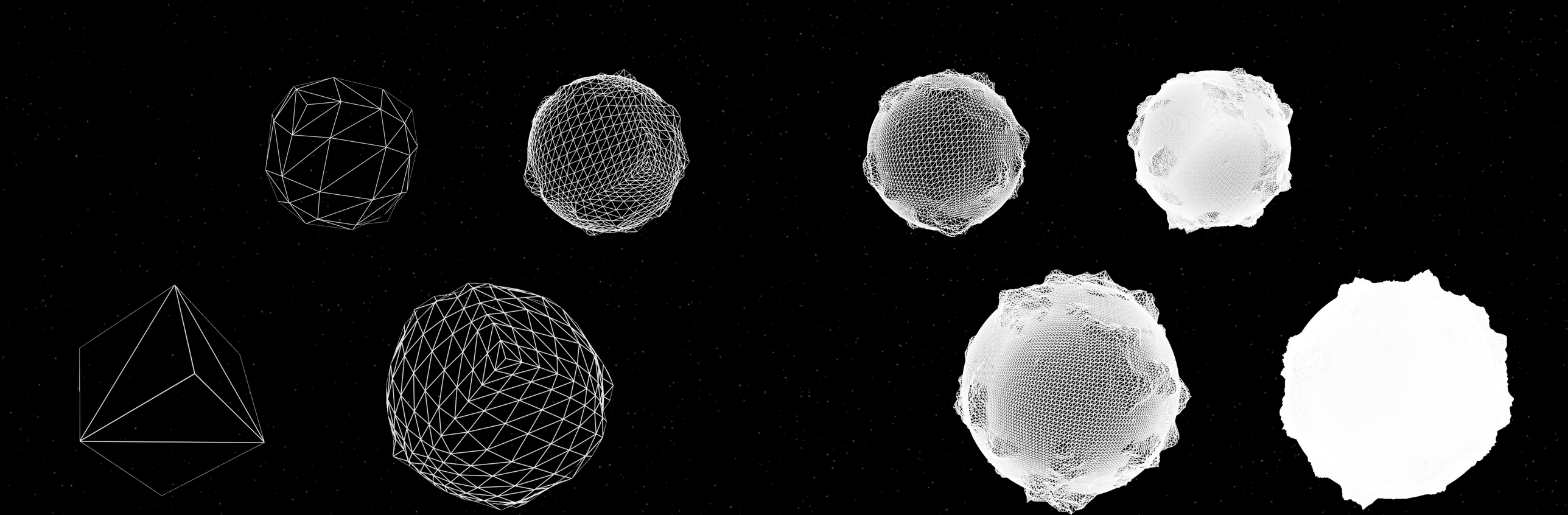


Abb. 44:·Planeten verschiedene Auflösungsstufen Collage 02, Unity Screenshots

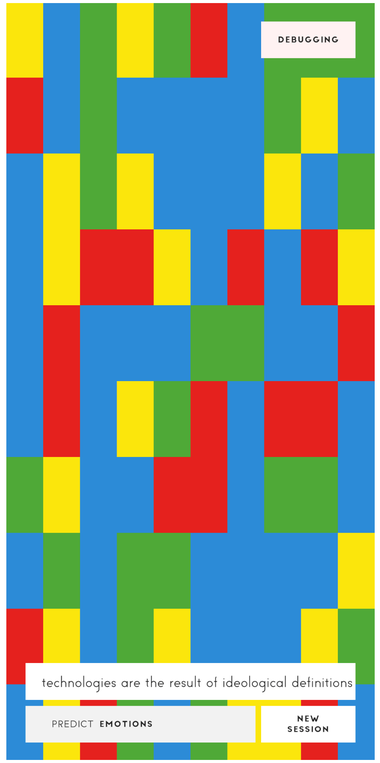


Abb. 45: Farben Prototyp 01 Visualisierungsbeispiel 01, Unity Screenshot

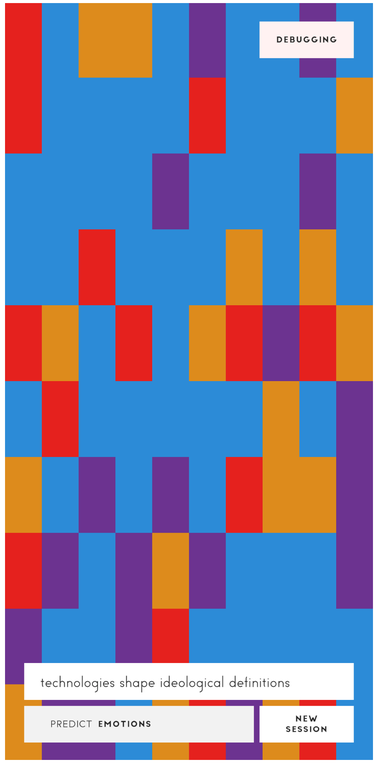


Abb. 46: Farben Prototyp 01 Visualisierungsbeispiel 02, Unity Screenshot

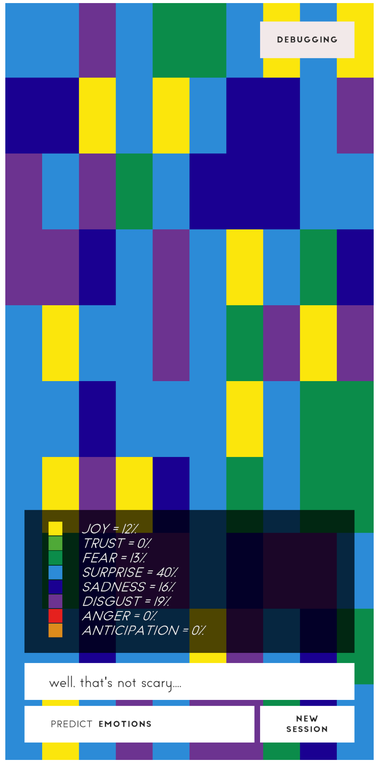


Abb. 47: Farben Prototyp 01 Visualisierungsbeispiel 03, Unity Screenshot

FARBEN Im Gegensatz zum Planetenansatz wurde der Farbenprototyp von Anfang an mit der Funktionalität der Emotionsberechnung verbunden. Auf der linken Seite ist die erste Iteration dieses Prototyps zu sehen. Das Smartphone ist weiterhin die Hauptplattform, da sich mit dem Wechsel der Visualisierung nicht zwangsläufig auch die grundlegende Herangehensweise für die Interaktion ändern musste. Auf den Screenshots sind 100 farbige Kacheln zu sehen, die jeweils einen Prozent der Vorhersage repräsentieren. Die Anordnung der farbigen Kacheln ist dabei randomisiert, um nicht als einfaches Balkendiagramm, sondern vielmehr als abstraktes Gefüge, als Bild wahrgenommen zu werden. Die hier aufgeführten Interaktionsmöglichkeiten beschränken sich auf ein Textinput Feld, einen Button zum Absenden des Textes, um eine Vorhersage auszulösen, einem Button, der die gesammelten Vorhersagen zurücksetzt und einem Debugging Button, der ein Overlay mit Prozentzahlen einblendet.

Die den acht Grundemotionen zugeordneten Farben sind den Farben aus Plutchiks oben erläuterten Wheel of Emotions entnommen. Ich habe mich an dieser Stelle auch der Farbenlehre zugewandt, um die Wirkung der Farben in diesem Prototypen weiter zu fundieren. Max J. Kobbert beschreibt in seinem Buch "Das Buch der Farben" eine Übung, die von Kunststudierenden in einem Hauptseminar durchgeführt wurde. Dort wurden Bilder zu bestimmten Gefühlen gemalt und anschließend verglichen und ausgewertet. Er schreibt diesbezüglich:

"Es kam zum Vorschein, dass Gefühle viel differenzierter und unterschiedlicher empfunden wurden, als es die Bezeichnungen erkennen lassen" (Kobbert 2019, S. 162). Um die historisch unterschiedlichen Herangehensweisen an das Farbverständnis besser zu einzuordnen, macht Kobbert die Unterscheidung zwischen der Wirkung und der Symbolik von Farben auf. Während ersteres mehr den Effekt beschreibt, den eine Farbe bei den Betrachter*innen auslöst, so sagt letzteres, die Symbolik, mehr über die autoritär gesetzten Zuordnungen und kulturell tradierten Bedeutungsmuster einer Farbe aus (vgl. Kobbert 2019, S. 162). Darüber hinaus schließen sich Theorien nach Goethes physikalischem Verständnis der Natur der Farben und zahlreiche zeitgenössische Interpretationen aus naturwissenschaftlicher, psychologischer und kunsttheoretischer Sicht an, die mich insgesamt zu dem Schluss kommen ließen, dass eine emotionsgestütze Einordnung von Farbbedeutungen nur innerhalb eines klar kommunizierten spezifischen Kontextes möglich ist. Demnach verhält es sich mit diesem Teil der Arbeit ähnlich, wie schon bei der Zuordnung von Emotionen zu Emojis:

Sofern die eindeutige Vermittlung der zugrunde liegenden Deep Learning Technologie im Vordergrund steht, müssen alle anderen Parameter für den jeweiligen Usecase konfigurierbar sein. Um in der beispielhaften Visualisierung konsistent zu bleiben, orientierte sich die Farbzuordnung also weiterhin an dem Modell von Robert Plutchik.

In der nächsten Version des Farbenprototyps wurden leichte User-Interface Anpassungen vorgenommen, die auf der rechten Seite zu sehen sind. Es wurde die Funktionalität hinzugefügt, die gesammelten Emotionswerte aller eingegebenen Nachrichten auf einmal zu sehen. Das Debugging Overlay führt diesbezüglich zwei Blöcke an Prozentwerten auf. Der Erste zeigt dabei die Prozentwerte für die letzte abgeschickte Nachricht, der Zweite die akkumulierten Werte aller Nachrichten. Wenn der darunter liegende Toggle mit der Bezeichnung "Enable Memory" aktiviert ist, wirken sich demgemäß alle Nachrichten auf die farbliche Visualisierung aus.

Ein weiterer Punkt, der mit dieser Prototypversion behandelt werden sollte, war die Interpretationsprozesse lebendiger zu gestalten und somit die Wahrnehmung der Mensch-Maschine Kommunikation in ein organischeres Licht zu rücken. Um das zu erzielen wurden die farbigen Kacheln in Bewegung versetzt. Mit dem Unity Plugin DOTween wurden randomisierte animierte Farbübergänge zwischen den Kacheln hergestellt, die an Ein- und Ausatmen erinnern sollten. Die Frequenz und Geschwindigkeit dieser Atmungsanimationen wurde dabei an die Unterschiedlichkeit der Texteingaben gekoppelt.

Konkret bedeutet das, dass wenn die Nutzer*innen überwiegend ähnliche, beispielsweise freudige Nachrichten eingegeben haben und nun eine aggressive, wütende Nachricht folgen lassen, so reagiert das System geschockt, färbt sich kurzfristig nahezu komplett zu der neuen, polarisierenden Emotion und blendet langsam wieder zu der gesammelten Ansicht aller eingegebenen Nachrichten zurück. Solche Verhaltensimplementierungen können dem schlichten Visualisierungssystem einen Charakter verleihen, der der Interaktion einen persönlicheren und konsequenzbehafteteren Kontext verleiht.

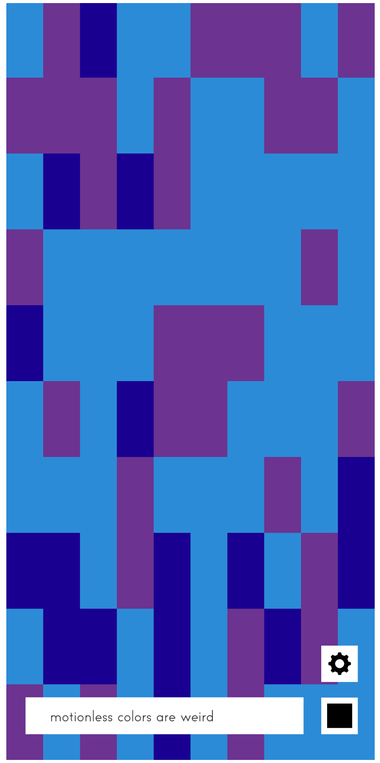


Abb. 48: Farben Prototyp 02 Visualisierungsbeispiel 01, Unity Screenshot

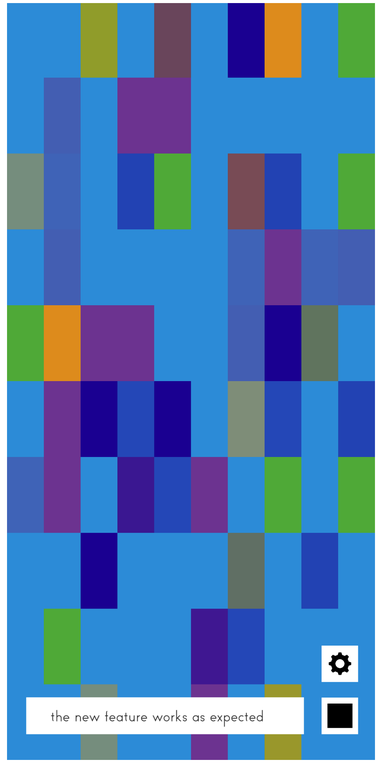


Abb. 49: Farben Prototyp 02 Visualisierungsbeispiel 02, Unity Screenshot

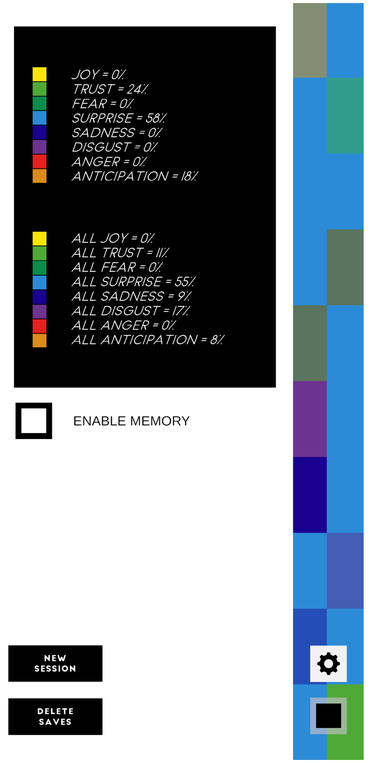


Abb. 50: Farben Prototyp 02 Visualisierungsbeispiel 03, Unity Screenshot

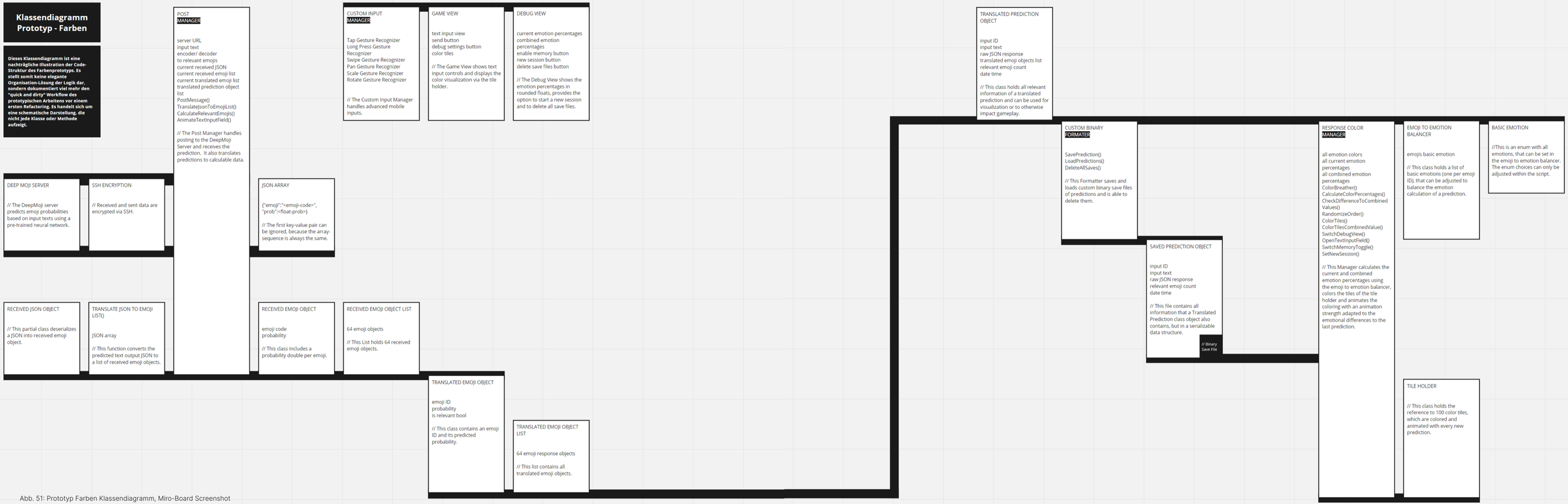


Abb. 51: Prototyp Farben Klassendiagramm, Miro-Board Screenshot

FARBEN FEEDBACK

Mit der Arbeit am Farbenprototypen gingen gelegentliche Playtests einher, die interessante Ergebnisse für die weitere Entwicklung und auch den anschließenden neuen Schwerpunkt des Designprojekts, das DeepMoji to Unity Programming Interface, bereithielten. Die zwei wichtigsten Erkenntnisse bezogen sich auf die Themen Choice Paralysis und Playfulness.

Choice Paralysis oder auch Overchoice oder Choice Overload sind kognitive Prozesseigenschaften, die beschreiben, wie sich ein Überangebot an Interaktionsmöglichkeiten verlangsamend oder gar lähmend auf den Entscheidungsprozess auswirken kann. Dieses Phänomen trat oftmals auf, wenn den Test-Nutzer*innen für die Texteingabe im vorliegenden Prototypen kein strukturgebender Kontext gegeben wurde. Dazu ist allerdings anzumerken, dass einige der Testenden den Bezugsrahmen eines digitalen Tagebuchs auch deswegen nicht zur Gänze adaptieren konnten, weil sie das Gefühl hatten bei der Eingabe beobachtet zu werden, und sich somit schlicht nicht trauten persönliche Nachrichten einzugeben.

Diese Testsituationen brachten aber auch zum Vorschein, dass die allgemeine Neugierde und die Herausforderung das Ironieverständnis des Modells zu testen allein, genug Antrieb war, um spielerisch mit der Applikation zu interagieren. Dabei musste allerdings auch immer wieder darauf hingewiesen werden, dass die Vorhersage mit längeren, ausformulierten Sätzen akkurater wird, was für einige Proband*innen zunächst kontraintuitiv erschien.

Abb. 52: Farben Prototyp 02 Visualisierungsbeispiel 04, Unity Screenshot

Diese spielerische Herangehensweise an den Umgang mit den Vorhersagen zog weitere Ideen der Inszenierung nach sich, die die beobachtete Playfullness und das ungefilterte Erleben mehr in den Mittelpunkt stellten.

Dazu gehörten unter anderem der Gedanke an eine untermalende Soundkulisse, deren Akkorde und Tempo sich durch die Emotionsvorhersagen beeinflussen lassen würden, aber auch Überlegungen, die Visualisierung der Emotionen über das Handydisplay hinaus wachsen zu lassen, und beispielsweise Projektoren oder Stagelights in die Wiedergabe miteinzubeziehen. Die Möglichkeit eine gemeinschaftlich erfahrbare Anwendung zu entwickeln kann dabei dem angestrebten Moment der Selbstwahrnehmung eine interessante Einordnung in die Gefühlswelt weiterer Teilnehmer*innen beifügen und die Kommunikation mit dem Algorithmus um eine zwischenmenschliche Kommunikationsebene ergänzen.

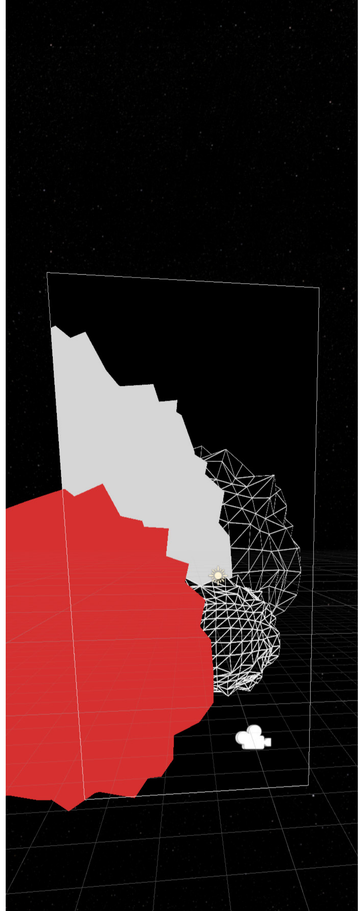


Abb. 53: Stencil Shader Tests 02, Unity Screenshot

MULTIUSER ERFAHRUNG

Eine sich an die, aus dem Feedback entwickelten Ansätze anschließende Überlegung, die sich an den Farbenprototyp orientiert hatte, war es, mit dem vorhandenen Visualisierungssystem einen kompletten Raum zu bespielen. Die bereits in der letzten Iteration hinzugefügte Möglichkeit Emotionswerte zu aggregieren und zwischen ihnen zu animieren würde in diesem Fall im Mittelpunkt der Erfahrung stehen.

Eingabegerät wäre in diesem Fall weiterhin ein Smartphone pro Nutzer*in, auf dem auch weiterhin nur die Emotionen der Nachrichten dargestellt werden, die die Nutzer*innen auch selbst eingegeben haben. Der mit Screens oder Projektionen ausgestattete Raum hingegen würde eine Sammlung der Emotionen aller Besucher*innen widerspiegeln, indem eine einfache, auf RPC calls basierende Verbindung zu allen sich im Netzwerk befindlichen Smartphone-Clienten genutzt wird.

Eine solche Installation könnte darüber hinaus auch Scheinwerfer via DMX ansteuern und eine dynamisch generierte Soundkulisse schaffen. Da ich mich mit den für solche Umsetzungen benötigten Plugins und APIs bereits in anderen Projekten auseinandergesetzt habe und sich die entsprechenden Tools oft durch hohe Verfügbarkeit und Einsteigerfreundlichkeit auszeichnen, habe ich mich aber entschlossen meinen Fokus mehr auf die, diesem Projekt eigenen, Besonderheiten zu legen, um anderen Entwickler*innen ebenfalls die Möglichkeit zu geben, interessante Software-Lösungen mit dem DeepMoji Framework schnell und einfach in Unity zu entwickeln.

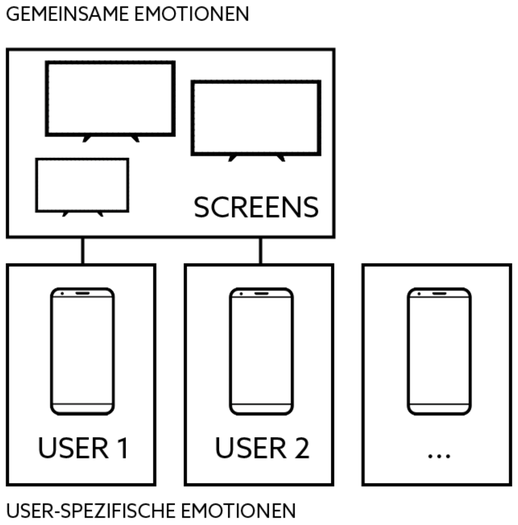


Abb. 54: Multi-User Installation Schaubild

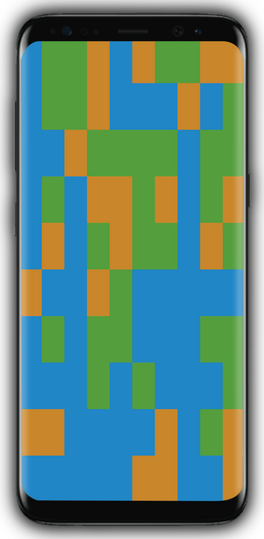


Abb. 55: Planeten Prototyp Visualisierungsbeispiel

Abb. 56: Formen Prototyp Visualisierungsbeispie

Abb. 57: Farben Prototyp 02 Visualisierungsbeispiel 05

ZWISCHENFAZIT

Bereits in meinen Beratungsgesprächen wurde mir der Hinweis gegeben mich mit dem vorliegenden Projekt mehr auf die Wiedernutzbarmachung zu konzentrieren und das Projektziel eher in einem nachhaltigen Tool zu sehen.

Die Ergebnisse des Prototyping haben auf unterschiedliche Weise in dieselbe Richtung gezeigt. Die Bestimmung, welche Emotionen zu welchen Emojis passen haben gleichermaßen aufgezeigt, wie kontextspezifisch eine adäquate Konfiguration sein kann, wie es auch die Interpretation der Farben gezeigt hat. Abhängig von der Zielgruppe, der kulturellen Konditionierung und der Intention der Applikation sind hier zahlreiche unterschiedliche Anordnungen, Zusammenstellungen und Abhängigkeitsbeziehungen denkbar. Der zuvor beschriebene Grundsatz, die hier entwickelte Applikation so transparent und unmittelbar wie möglich zu gestalten, hat die Bedeutung dieser rahmengebenden Eigenschaften immer wieder dialektisch begründet.

Somit ist es nur konsequent, diese bisher als extern behandelten Einflussfaktoren zu entscheidenden Konfigurationsoptionen eines DeepMoji to Unity Programming Interfaces (DM2UPI) zu machen und bis zu diesem Zeitpunkt angesammelte, explorative Ergebnisse in die Entwicklung einer eigenen API einfließen zu lassen.

Die Fokussierung auf ein solches, konkretes Produkt erlaubt es nun auch aus dem prototypischen Workflow in die explizite Produktion überzugehen und die Softwarearchitektur anhand von zuvor dokumentierten, aus dem Prototyping hervorgegangenen Entwickler*innen-Bedürfnissen neu zu strukturieren.

DM2UPI

Das DeepMoji to Unity Programming Interface (DM2UPI) soll die Möglichkeit bieten unterschiedlichste Projekte mit der Funktionalität des Emotional Understandings auszustatten. Dazu gehören neben den von mir bereits angeführten Ansätzen eines Emotionen visualisierenden Tagebuchs, oder emotionsvisualisierungsfundierten Installationen auch andere denkbare Ausstellungssituationen und nicht zuletzt die Erweiterung klassischer Spielmechanismen und -elemente, wie beispielsweise adäquat reagierende Gesichtsanimationen von Spielcharakteren. Die Gestaltung von digitalen Spielen ist immer an den Umfang und für Indi- und Universitätsprojekte auch an die Erschwinglichkeit und Zugangsbedingungen von Entwicklungstools gebunden, die entweder als Spielelemente dienen können, oder deren Entwicklung begünstigen. Wie eingangs erwähnt sind die monetären Zugangsbeschränkungen zu Deep Learning Modellen im Bereich Emotional Recognition und Understanding noch sehr hoch, und so hoffe ich mit meiner API einen kleinen Beitrag zur Demokratisierung dieser Technologie beitragen zu können.

Es motiviert mich im Besonderen, die durch das Prototyping erlangten Erkenntnisse und Software nicht nur selbst für Folgeprojekte nutzen zu können, sondern im gleichen Maße anderen Entwickler*innen zur Verfügung zu stellen. Wie ich bereits während diesem Designprojekt auf viele unterschiedliche Open Source Quellen zugreifen konnte, sollten auch andere Entwickler*innen von meiner Arbeit profitieren können, die ohne die Veröffentlichungen des MIT und einer aufgeschlossenen Online-Community aus findigen Entwickler*innen so nicht möglich gewesen wäre.

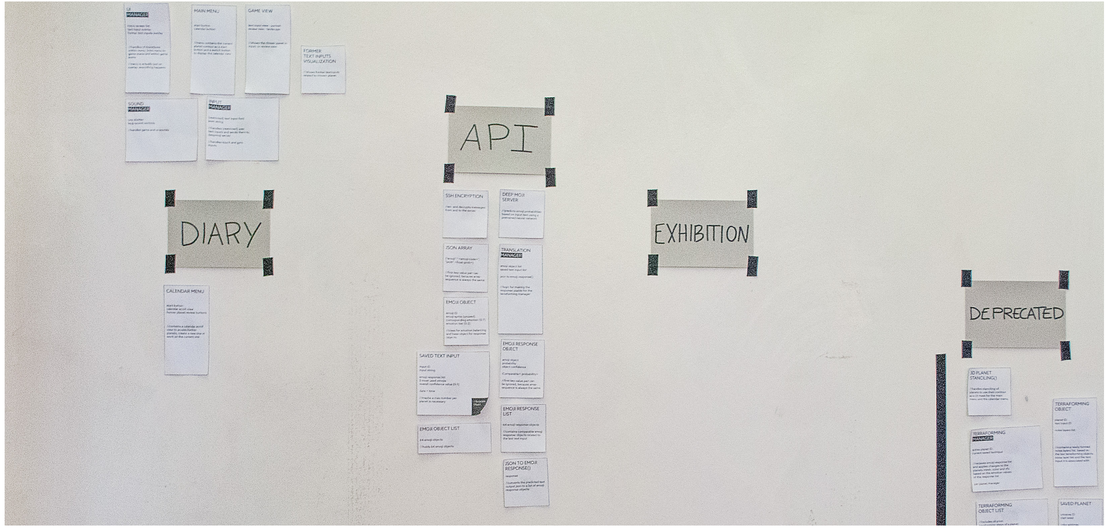


Abb. 01: Abb. 58: Analoges Prototyping für Klassendiagramm 03

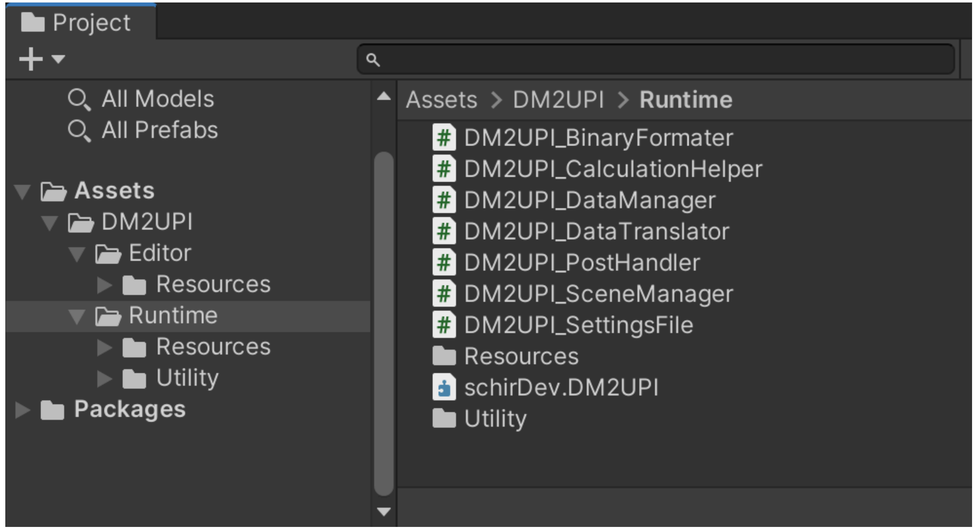


Abb. 59: DM2UPI Projektordner, Unity Screenshot

ZIELGRUPPE UND BEDARFE

Die API richtet sich direkt an Unityentwickler*innen, die Emotional Understanding für ein eigenes Projekt nutzen wollen. Damit gibt es eine klare Zielgruppe, deren Bedarfe es vor einem Code-Refactoring abzuklären gilt. Eigenschaften, die diese API mit sich bringen sollte, sind:

- **hohe Wiederverwendbarkeit**
- **Ausbaufähigkeit**
- **Nutzung vertrauter Development-Schnittstellen**
- **schnelle Prototyping Iterationen**

Dabei sollte sich das von den Entwickler*innen benötigte Verständnis des zugrunde liegenden Deep Learning Frameworks in Grenzen halten. Die notwendigen Schnittstellen sollten auf ein Minimum reduziert werden. Notwendig ist in diesem Falle die Möglichkeit Konfigurationen vorzunehmen, die bestimmen, wie die DeepMoji Vorhersagen interpretiert werden und die Möglichkeit vorstrukturierte Auswertungsklassenelemente zu empfangen, die alle relevanten, berechneten Informationen enthalten, die benötigt werden, um für ein aufbauendes Projekt genutzt zu werden. Die genannte Konfigurationsmöglichkeit sollte übersichtlich gestaltet sein, und entgegen den hard-gecodeten Emotionsbezeichnungen der bisherigen Prototypen ein flexibles System zur Verfügung stellen, das unterschiedliche Emotionsdefinitionen und Zuordnungen für jedes neue Projekt ermöglicht und einfach abspeicherbar und übertragbar macht. Gleichzeitig sollten Emojis eine unbegrenzte Anzahl von Emotionen zugeordnet werden können, um das Setup so feingranular wie gewünscht machen zu können, ohne dass dies Komplikationen in der Berechnung nach sich zieht oder Code-Anpassungen erforderlich macht. Diese unbestimmte Anzahl an Zuordnungen sollte zugleich untereinander gewichtet werden können, um auch ungleichwertige Emotionsverteilungen in einem Emoji nachstellen zu können.

Das System soll weiterhin auf einer Server-Client Struktur basieren. Die Vorteile dieses Ansatzes wurden zu Beginn dieser Arbeit bereits aufgeführt.

Der Code sollte strukturiert, nachvollziehbar, kommentiert und entkoppelt sein, er sollte eine hohe Lesbarkeit aufweisen und eindeutige Benennungen nutzen. Die Zugriffsmodifizierer sollten so gewählt sein, dass andere Entwickler*innen keine unnötigen Berechtigungen für Klassen erhalten, die nicht für die Modifikation oder Interaktion durch Endnutzende gedacht sind. Dies soll auf der einen Seite mögliche Fehlerquellen reduzieren, auf der anderen Seite auch zur Übersichtlichkeit beitragen, da somit bei der Nutzung der DM2UPI Library nur Klassen über das Autocomplete-Feature des Code-Editors vorgeschlagen werden, die auch eine Relevanz für die Entwickler*innen haben. Folgerichtig sollten somit auch alle Codeelemente, die keine Nutzer*inneninteraktion voraussetzten, entweder automatisiert instanziiert werden, oder als statische Klassen und/ oder Methoden implementiert sein.

Neben der umfangreichen Funktionalitäten der API, sollte es ein angemessenes User Interface geben, dass die Interaktion im Editormodus vereinfacht.

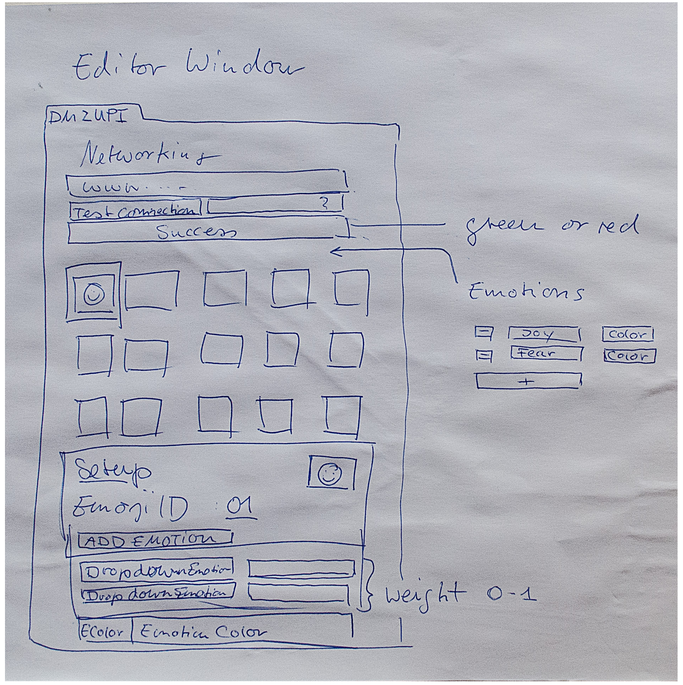


Abb. 60: Settings Editor Lernjournal Skizze

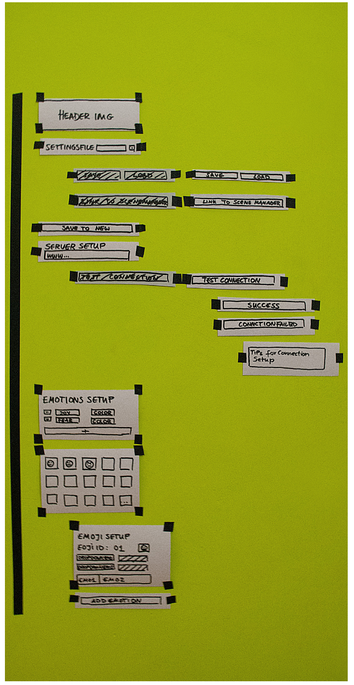


Abb. 61: Arbeitssituation 02 Settings Editor Pen & Paper Prototyping

SETTINGS EDITOR

Der Settings Editor ist eine von zwei direkten Schnittstellen zu den Nutzer*innen der API. Hier kann die rechts zu sehende Settingsfile konfiguriert werden. Das Settings Editor Window ist im Unity Header Menü integriert und kann von dort geöffnet werden. Grundsätzlich ist das Settings Editor Window eine User Interface Strukturierung, die für alle Parameter der Settingsfile übersichtliche Einstellungsoptionen bietet und jederzeit revidierbar ist.

Auf der vorherigen Seite sind eine Skizze und ein Pen and Paper Prototyp des Userinterfaces zu sehen. Diese habe ich vor der Entwicklung des Settingseditors angefertigt, um eine klare Richtlinie für die Entwicklung zu haben. Angefangen von einer Objektreferenz zur Settings File, die aktuell bearbeitet wird finden sich dort eine Netzwerkkonfiguration, mit der Möglichkeit die Serververbindung auf Knopfdruck zu testen, ein Informationsfeld, dass Elemente wie eine Settings File oder einen Scene Manager auflistet, wenn diese fehlen sollten und Emotions-, wie Emoji-Konfigurationsfelder. Diese Konfigurationsfelder stehen in direkter Abhängigkeit zueinander. So werden in der Emojikonfiguration nur Emotionen angezeigt, die zuvor auch unter “Emotions” angelegt wurden. Sollten Emotionen wieder entfernt werden, so werden alle Emojis und ihre Gewichtungswerte automatisch angepasst, die nun fehlende Emotion entfernt und ihr prozentualer Anteil an bereits konfigurierten Emojis den restlichen dort jeweils zugeordneten Emotionen beigefügt. Die Gewichtung der Emotionen innerhalb und bezüglich eines Emojis kann durch die Nutzung eines Multi-Value-Sliders vorgenommen werden, der immer genauso viele Elemente enthält, wie es zugeordnete Emotionen zu einem Emoji gibt.

Da sich all diese Funktionalitäten im Bereich des Editorscripting abspielen, ist ein näherer Einblick später zu finden, wenn auf die Gestaltung des Custom-Editor-Windows eingegangen wird.

SETTINGS FILE

Die Settings File beinhaltet alle für Vorhersage und deren Auswertung relevanten Einstellungen, die im Settings Editor konfiguriert werden können. Es handelt sich dabei um ein Scriptable Objekt, dass einfach über das Kontextmenü im Projektordner instanziiert werden kann. Der Vorteil dieser Herangehensweise ist, dass eine Settings File somit einfach von Projekt zu Projekt und von Szene zu Szene übertragen und genutzt werden kann, ohne dass es notwendig ist den Konfigurationsvorgang zu wiederholen. Eine Settings File kann so auch als Grundlage für leichte Änderungen und Anpassungen genutzt werden, da sie sich als Scriptable Objekt einfach kopieren und bearbeiten lässt. Das Objekt kann auch direkt im Inspektor bearbeitet werden, da diese Herangehensweise aber wenig Übersicht bietet und bei der Fülle an zu konfigurierenden Einstellungen leicht zu Flüchtigkeitsfehlern führen kann, ist eine Bearbeitung über den Settings Editor pragmatischer.

Neben den Emotions- und Emojieinstellungen sind in der Settingsfile auch die Serverreferenz und der emojiPredictioncount, sowie der emojiPredictionTreshold zu finden. Die Serveradresse gibt an über welche Webadresse der zu nutzende DeepMoji Server erreichbar ist. Der emojiPredictionCount und emojiPredictionTreshold sind Variablen, von denen jeweils nur die eine oder die andere für die Vorhersage genutzt werden. Entweder wird mit dem emojiPredictionCount eine feste Anzahl von Emojis festgelegt, die Einfluss auf die Vorhersage haben sollen, oder es wird über den emojiPrediction-Treshold ein minimaler Prozentwert definiert, den ein Emoji überschreiten muss, um in die Vorhersage mit einzufließen. Welche der Variablen genutzt wird, wird im Scene Manager angegeben. Da eine Vorhersage des DeepMoji Modells jedem einzelnen Emoji eine, wenn auch teilweise unter einem Prozent liegende Wahrscheinlichkeit zuweist, kann es sinnvoll sein an dieser Stelle eine Beschränkung vorzunehmen.

```
/*
 * schirDev.DM2UPI - DeepMoji to Unity Programming Interface
 * by Caspar Schirdewahn - Game Design B.A. - HTW Berlin
 */

using System.Collections.Generic;
using UnityEngine;

namespace DM2UPI
{
    namespace Settings
    {
        /// <summary>
        /// This ScriptableObject contains the configuration used to connect to the deep moji server,
        /// choose the right emojis and calculate the emotion percentages.
        /// </summary>
        [CreateAssetMenu(fileName = "New DM2UPI Settings File", menuName = "DM2UPI Settings File")]
        6 references
        public class DM2UPI_SettingsFile : ScriptableObject
        {
            [SerializeField]
            public string serverAdress = "https://...";
            [SerializeField]
            public List<DM2UPI_Emotion> emotionRange = new List<DM2UPI_Emotion>();
            [SerializeField]
            public DM2UPI_EmotionSetup[] emojiIDSetup = new DM2UPI_EmotionSetup[64];

            [Range(0, 64)]
            public int emojiPredictionCount = 5;
            [Range(0, 1)]
            public float emojiPredictionThreshold = 0.1f;
        }

        [System.Serializable]
        10 references
        public class DM2UPI_Emotion
        {
            public string description;
            public Color color;

            /// <summary>
            /// This object is used to setup a new emotion.
            /// </summary>
            /// <param name="newDescription">written description of the emotion</param>
            /// <param name="newColor">corresponding color to the emotion</param>
            3 references
            public DM2UPI_Emotion(string newDescription, Color newColor)
            {
                description = newDescription;
                color = newColor;
            }
        }

        [System.Serializable]
        8 references
        public class EmotionBalance
        {
            public DM2UPI_Emotion emotion = new DM2UPI_Emotion("", Color.white);
            public float proportion;

            /// <summary>
            /// This object contains a DM2UPI_Emotion and the proportion it has compared to
            /// other DM2UPI_Emotion's linked to the same emoji.
            /// </summary>
            /// <param name="newEmotion">the DM2UPI_Emotion object</param>
            /// <param name="newProportion">the proportion that this emotion will impact the percentageShare calculation</param>
            2 references
            public EmotionBalance(DM2UPI_Emotion newEmotion, float newProportion)
            {
                emotion = newEmotion;
                proportion = newProportion;
            }
        }

```

Abb. 62: DM2UPI Settings File Code, VisualStudio Screenshot

```
#region Predict
/// <summary>
/// Predicts Emotions of the textToPredict.
/// </summary>
/// <param name="textToPredict">the string the Emotions are predicted for</param>
/// <param name="saveBinary">should this prediction be saved as a binary file?</param>
0 references
public void PredictEmotions(string textToPredict, bool saveBinary)
{
    if (settingsFile != null)
        DM2UPI_PostHandler.PredictEmojis(settingsFile.serverAdress, textToPredict, saveBinary);
}

/// <summary>
/// Predicts Emotions of the debugTextInput of the SceneManager.
/// SceneManagers savePredictionsDefault determins if the
/// prediction will be saved as a binary file.
/// </summary>
1 reference
public void PredictEmotions()
{
    if (settingsFile != null)
        DM2UPI_PostHandler.PredictEmojis(settingsFile.serverAdress, debugTextInput, savePredictionsDefault);
}

/// <summary>
/// Predicts Emotions of the textToPredict.
/// SceneManagers savePredictionsDefault determins if the
/// prediction will be saved as a binary file.
/// </summary>
0 references
public void PredictEmotions(string textToPredict)
{
    if (settingsFile != null)
        DM2UPI_PostHandler.PredictEmojis(settingsFile.serverAdress, textToPredict, savePredictionsDefault);
}


#region Save
#region Load
#region Unload
#region Delete
#region Events
2 references
private void OnNewTranslatedPredictionDataSaved(DM2UPI_TranslatedPrediction translatedPrediction)
{
    if (logNextPrediction)
        LogPredictionOutput(translatedPrediction);

    if (translatedPrediction.toBeSaved)
        SavePrediction(translatedPrediction);
    else
        RefreshDataReferences();

    if (NewPredictionAvailableEvent != null)
        NewPredictionAvailableEvent.Invoke(translatedPrediction);
}
#endregion
9 references
private void RefreshDataReferences()...
```

Abb. 63: DM2UPI Scene Manager Code Collage, VisualStudio

```
using System;
using System.Collections;
using UnityEngine;
using UnityEngine.Networking;

[assembly: System.Runtime.CompilerServices.InternalsVisibleTo("schirDev.DM2UPI.Editor")]

namespace DM2UPI
{
    namespace Networking
    {
        10 references
        internal static class DM2UPI_PostHandler
        {


2 references
private static UnityWebRequest PrepareWebRequest(string serverAdress, string message)
{
    string jsonContent = "{\"sentences\": [\"" + message + "\"]}";
    byte[] jsonPost = new System.Text.UTF8Encoding().GetBytes(jsonContent);

    UnityWebRequest request = new UnityWebRequest(serverAdress, "POST");
    request.uploadHandler = (UploadHandler)new UploadHandlerRaw(jsonPost);
    request.downloadHandler = (DownloadHandler)new DownloadHandlerBuffer();

    return request;
}


1 reference
private static IEnumerator PostMessage(UnityWebRequest request, string message,bool savePrediction)
{
    yield return request.SendWebRequest();

    if (request.isNetworkError)
    {
        Debug.LogWarning("DM2UPI WebRequest produced the following Error: " + request.error);
    }
    else
    {
        DM2UPI_DataHandler.current_saveBinary = savePrediction;
        DM2UPI_DataHandler._current_InputText = message;
        DM2UPI_DataHandler.current_DM_Prediction = request.downloadHandler.text;
        DM_PredictionReceived();
    }
```

Abb. 64: DM2UPI Post Handler Code Collage, VisualStudio

SCENE MANAGER

Der Scene Manager bildet die zweite direkte Nutzer*innenschnittstelle für Entwickler*innen. Hier laufen alle Funktionalitäten der API zusammen. Er beinhaltet Methoden, um die Vorhersage zu starten, zu speichern, zu laden, zu entladen und zu löschen. Je nach Einstellung wird hierfür der DM2UPI Binary Formatter genutzt. Alle Methoden sind mit Überladungen ausgestattet, die unterschiedliche Möglichkeiten für die Texteingabe bereitstellen und globale wie lokale Referenzen für die Nutzung der Speicherfunktionalität erlauben. Es gibt ein verfügbares Unity Event, das ausgelöst wird, sobald eine neue Vorhersage empfangen und vom System verarbeitet und aufbereitet wurde. Diese Translated Prediction Klasse wird mit dem Unity Event an alle registrierten Methoden übergeben. Ich habe mich an dieser Stelle dazu entschieden das Unity Event System zu nutzen, da es die Möglichkeit bietet, neben der Zuschreibung von Funktionen via Code, auch Zuschreibungen über den Inspektor zu erlauben, was die Nutzbarkeit der API für Personen ohne große Coding-Kenntnisse erleichtert.

Der Manager ist als Singleton angelegt, der über ein Singleton Management System threadsicher implementiert ist. Da die Hintergrundprozesse der API aufgrund des entkoppelten Server-Client Systems asynchron zum Main Thread der aktuellen Unity Szene ablaufen, könnte es ansonsten zu fehlerhaften Referenzierungen kommen. Um das Singleton Pattern nicht überzustrapazieren ist der Scene Manager der einzige nutzer*innenseitige Singleton in dieser API.

Im Inspektor kann man zwischen den in der Settings File angelegten topEmojiPriority Werten (emojiPredictionCount oder emojiPredictionTreshold) wechseln, per Knopfdruck gespeicherte Vorhersagen laden, entladen oder löschen, die letzte und alle bisherigen Vorhersagen einsehen und über das Debugging Fenster Vorhersagen in der Konsole drucken.

POST HANDLER

Der Post Handler regelt alle Netzwerkfunktionalitäten der API, die mit der Interaktion zwischen der Unity Szene und dem DeepMoji Server zu tun haben. Die unityseitig verwendeten Frameworks wurden bereits im Kapitel “Unity Networking” behandelt. An dieser Stelle sind nun noch die API bezogenen Besonderheiten zu nennen. Um die Grundfunktionalitäten der API, wie bereits angeführt automatisiert und ohne Instanziierungsnotwendigkeit durch die Nutzer*innen ablaufen zu lassen, ist der Post Handler als statische Klasse angelegt, die jeder Zeit vom Scene Manager angesprochen werden kann. Somit muss weder eine eigene Komponente in die Szene hinzugefügt werden, noch müssen Funktionalitäten im Scene Manager angesammelt werden, die nicht zu seinem Aufgabenbereich gehören und somit der angestrebten Entkoppelung des Codes entgegenstehen. Die Klasse als statisch zu deklarieren brachte die Problematik mit sich, dass für die hier benutzten Unity-Webrequests Coroutinen benutzt werden müssen, die nur auf instanziierten Klassen funktionieren. Diesbezüglich habe ich mich von der Unity Forum Nutzer*in CykesDev inspirieren lassen und eine Static Coroutine Klasse angelegt, die zu diesem Zweck Coroutinen Objekte instanziiert und nach Benutzung wieder entfernt.

Da eine Nutzung des Post Handlers nur über den Scene Manager vorgesehen ist, habe ich diese Klasse, wie auch andere Teile der Hintergrundabläufe der API mit dem “internal” Zugriffsmodifizierer versehen, der einen Zugriff nur für Klassen in derselben Assembly erlaubt. Um diese Einschränkung wirksam zu machen habe ich sowohl für den Runtime- als auch den Editorteil der API eine eigene Assembly angelegt und konfiguriert.

Alle empfangenen Vorhersagen des Servers werden in den DM2UPI Data Handler übertragen und dort mitsamt des Eingabetextes gespeichert. Dort wird anschließend die weitere Datenverarbeitung ausgelöst.

```
namespace DM2UPI
{
    16 references
    internal static class DM2UPI_DataHandler
    {
        private static bool m_current_saveBinary;
        private static string m_current_InputText;
        private static string m_current_DM_Prediction;

        private static DM2UPI_TranslatedPrediction m_current_Translated_Prediction;
        private static List<DM2UPI_TranslatedPrediction> m_session_Translated_Predictions = new List<DM2UPI_TranslatedPrediction>();

        1 reference
        internal static bool current_saveBinary...

        1 reference
        internal static string _current_InputText...

        1 reference
        internal static string current_DM_Prediction
        {
            get { return m_current_DM_Prediction; }
            set
            {
                m_current_DM_Prediction = value;
                DM_NewPredictionAvailable(m_current_InputText, m_current_DM_Prediction, m_current_saveBinary);
            }
        }

        4 references
        internal static DM2UPI_TranslatedPrediction current_Translated_Prediction
        {
            get { return m_current_Translated_Prediction; }
            set
            {
                m_current_Translated_Prediction = value;
                m_session_Translated_Predictions.Add(value);
                DM2UPI_NewTranslatedPredictionAvailable(m_current_Translated_Prediction);
            }
        }

        2 references
        internal static List<DM2UPI_TranslatedPrediction> session_Translated_Predictions
        {
            get { return m_session_Translated_Predictions; }
        }

        1 reference
        internal static bool RemoveSessionTranslatedPrediction(int index)...

        1 reference
        internal static bool RemoveSessionTranslatedPrediction(string filePath)...

        1 reference
        internal static void ResetSessionTranslatedPredictions()...

        internal static event Action<string, string, bool> DM_NewPredictionAvailable = delegate { };
        internal static event Action<DM2UPI_TranslatedPrediction> DM2UPI_NewTranslatedPredictionAvailable = delegate { };
    }
}
```

Abb. 65: DM2UPI Data Handler Code, VisualStudio Screenshot

DATA HANDLER

Sobald im Data Handler neue Vorhersagedaten durch den Post Handler eingetragen werden, werden diese Daten über C# Events an den DM2UPI Data Translator weitergegeben. Da auch der Data Translator nicht für die direkte Nutzer*innen-Interaktion angelegt ist, habe ich mich hier für die schnelleren und weniger Arbeitsspeicher benötigenden C# Events entschieden. Unity Events bieten neben der Möglichkeit Empfängermethoden im Inspektor zuzuweisen auch den Vorteil schwächere Referenzen zu nutzen, die bei der Zerstörung von Unity Objekten direkt freigegeben werden. Bei der Nutzung von Eventsystemen in statischen Klassen ist auf der anderen Seite gründlich darauf zu achten, dass zugewiesene Delegates auch wieder entfernt werden. Da statische Klassen in ihrer Nutzung weder instanziiert noch zerstört werden, ist dieser Umstand mit besonderer Sorgfältigkeit zu behandeln, da sonst unerwünschte Memoryleaks entstehen können. Der Data Handler ist in dieser API Teil eines abgeschlossenen Systems, es kommen also keine neuen oder unerwarteten Abonnenten zu den implementierten Eventsystemen hinzu, die nicht an die Nutzer*innen gerichtet sind. Somit sollte dieses Problem nicht auftreten.

So bald die Daten in, durch die Nutzer*innen verwendbare, Translated Prediction Klassen übersetzt, und wieder in den Data Handler übertragen wurden, werden diese Daten umgehend via C# Event an den Scene Manager gesendet, der diese dann an die Abonnenten-Methoden der Nutzer*innen weiterleitet. Darüber hinaus beinhaltet der Data Handler noch Funktionen für das Entladen von geladenen Translated Predictions und fungiert somit generell als zentraler Speicherort für alle Vorhersagen einer laufenden Sitzung.

BINARY FORMATTER

Der Binary Formatter wird vom Scene Manger genutzt, um aus empfangenen Translated Prediction Klassen Saved Prediction Dateien zu erstellen und für die spätere Nutzung zu speichern. Diese Saved Prediction Klassen können wiederum vom Data Translator in Translated Predictions zurückübersetzt werden. Eine Saved Prediction beinhaltet dabei die gleichen Informationen, wie eine Translated Prediction, allerdings sind diese Informationen in serialisierbaren Datentypen untergebracht.

Ob eine Datei gespeichert wird, wird bereits mit dem Absenden des für die Vorhersage bestimmten Textes angegeben und somit durch den gesamten Hintergrundprozess weitergeleitet. Dadurch wird die Verwendung einer globalen Variable umgangen, damit sich die asynchronen Vorhersage- und Verarbeitungsprozesse nicht mit der Setzung eines globalen Bools derart überschneiden, dass Informationen falsch zugeordnet werden könnten. Als Speicherpfad wird Unitys persistentDataPath genutzt, der plattformspezifisch immer auf denselben Ort auf dem jeweiligen Speichermedium verweist. Dateien in diesem Verzeichnis werden nicht durch Applikations-Updates überschrieben und stellen somit einen kohärenten kontrollierbaren Speicherort dar.

Neben den gespeicherten Dateien wird zusätzlich eine Referenzdatei angelegt, die die automatisierten Dateinamen und deren konkrete Adressierung beinhaltet. Diese Datei ist hauptsächlich dazu da, um die Speicherorte direkt im Scene Manager anzeigen lassen zu können und zweckgemäße, kopierbare Informationen für die Nutzer*innen bereitzustellen, die auch dafür genutzt werden können einzelne Vorhersagen bei Bedarf aus der Sitzung zu entladen oder direkt zu löschen.


```
internal static class DM2UPI_BinaryFormatter
{
    1 reference
    internal static void SavePrediction(DM2UPI_SavedPrediction savedPrediction, int sessionIndex)...

    0 references
    internal static List<DM2UPI_SavedPrediction> LoadPredictions()...

    3 references
    internal static DM2UPI_SavedPrediction LoadPrediction(string filePath)
    {
        string[] fileNames = LoadFileNamesReference();

        if (fileNames != null)
        {
            BinaryFormatter binaryFormatter = new BinaryFormatter();
            DM2UPI_SavedPrediction loadedPrediction;

            for (int i = 0; i < fileNames.Length; i++)
            {
                if (fileNames[i] == filePath)
                {
                    FileStream stream = new FileStream(fileNames[i], FileMode.Open);

                    DM2UPI_SavedPrediction data = binaryFormatter.Deserialize(stream) as DM2UPI_SavedPrediction;
                    stream.Close();
                    loadedPrediction = new DM2UPI_SavedPrediction(data);
                    return loadedPrediction;
                }
            }
            Debug.LogWarning("Tried to load File at FilePath: " + filePath + ", but the File does not exist.");
            return null;
        }
        else
        {
            Debug.LogWarning("Tried to load File at FilePath: " + filePath + ", but the File does not exist.");
            return null;
        }
    }

    2 references
    internal static bool SaveFileExists(string filePath)...

    3 references
    internal static List<string> GetSaveFileLocations()...

    1 reference
    private static int SavePredictionsCounter()...

    2 references
    private static int LoadPredictionsCounter()...

    1 reference
    private static void CreateNewPredictionCounter()...


    1 reference
    private static void SaveFileNamesReference(string fileNameReference)...

    1 reference
    private static void SaveFileNamesReference(string[] fileNamesReference)...

    7 references
    private static string[] LoadFileNamesReference()...
    1 reference
    private static string[] CreateNewFileNamesReference(string fileNameReference)...

    1 reference
    internal static void DeleteSave(string filePath)...
```

Abb. 66: DM2UPI Binary Formatter Code, VisualStudio Screenshot

DATA TRANSLATOR

Der Data Translator übernimmt alle Aufgaben, die erfordern, einen bestimmten Datentyp in einen anderen zu übertragen. Das umfasst die Auswertung eines vom DeepMoji Server empfangenen JSON Files, genauso wie die Übersetzung einer Saved Prediction in eine Translated Prediction und umgekehrt. Um alle, in einer Translated Prediction aufgeführten, Informationen zu berechnen, werden die decodierten JSON Daten an den DM2UPI Calculation Helper weitergegeben. Diese Klasse führt alle Rechenoperationen durch, die notwendig sind, um die gesammelten Prozentwerte nach Emotionen geordnet und anhand der, in der Setings File gewählten, Count- oder Threshold-Einstellung, verdichteten Relevanz auszugeben. Der Data Translator beinhaltet ebenfalls alle serialisierbaren Klassen, die für die Strukturierung der Vorhersagen gebraucht werden.

Zum Parsing der eintreffenden JSON Dateien wird hier Newtonsofts open source Framework Json.NET benutzt. Dieses Framework ist genau wie das DeepMoji Modell unter der MIT Lizenz verfügbar und somit für die Nutzung auch in kommerziellen Projekten freigegeben.

Da der Data Translator mit instanziierten Objekten arbeitet, ist er der zweite und letzte Singleton in der API. Er ist nicht für die Nutzung durch Entwickler*innen vorgesehen und wird daher automatisch an den Scene Manager angehängt. Der Zugriff ist wie bei allen Klassen, die den funktionalen Unterbau dieser API bilden durch den “internal” Zugriffsmodifizierer beschränkt. Seine Funktionalitäten werden dementsprechend nur automatisiert durch das C# Eventsystem und direkte interne Referenzen ausgelöst und verwendet.

TRANSLATED PREDICTION

Die Translated Prediction stellt als Datensammelpunkt die dritte und letzte Interaktionsschnittstelle zu Nutzer*innen dar. Diese Klassenobjekte werden vom Data Translator erstellt und durch den Scene Manager verteilt.

Im Kapitel “Technische Grundlagen” wurden bereits JSON Objekte aufgezeigt, die als Ausgabe des DeepMoji Servers entstehen. In einer Translated Prediction finden sich nunmehr alle daraus kalkulierten Informationen und weitere nützliche Variablen.

Dazu gehören zum Beispiel zwei einzigartige IDs; die predictionID und die objectID. Während die predictionID an die zugrunde liegende Vorhersage gekoppelt ist, sich also auch beim Speichern und Laden nicht verändert, ist die objectID bei jedem Ladevorgang eine neue. Die Kombination beider IDs macht es den Nutzer*innen also möglich, auch beim Nachladen von Vorhersagen immer klar definieren zu können, ob sie eine bestimmte Vorhersage referenzieren möchten oder das entsprechende nachgeladene Objekt.

Weitere nützliche Informationen, die sich in der Translated Prediction finden, sind der ursprüngliche Inputtext, Datum und Zeit der abgeschickten Anfrage, die Anzahl an Emojis, die Auswirkung auf die Prozentberechnung hatten und die IDs der Emojis, die in die Prozentberechnung miteingeflossen sind. Hauptinhalt ist aber der predictedEmotionShare, in dem die Emotionen und ihre prozentuale Verteilung aggregiert aufgeführt sind, die in den, als relevant behandelten, Emojis konfiguriert waren. Hier lassen sich jetzt die auf 100 % verteilten berechneten Emotionswerte ablesen.


```
2 references
private void OnNewPredictionAvailable(string newInputText, string newPrediction, bool saveBinary)
{
    DM2UPI_DataHandler.current_Translated_Prediction = TranslateFromJson(newInputText, newPrediction, saveBinary);
}

2 references
internal DM2UPI_TranslatedPrediction TranslateFromJson(string inputText, string jsonString, bool saveBinary)
{
    ReceivedJsonObject jsonObject = ReceivedJsonObject.FromJson(jsonString);

    List<ReceivedEmojiObject> emojiList = new List<ReceivedEmojiObject>();
    List<DM2UPI_EmojiIDObject> newPredictedEmojiIDObjects = new List<DM2UPI_EmojiIDObject>();

    emojiList = jsonObject.SentencesPredictionList[0];
    newPredictedEmojiIDObjects = DM2UPI_CalculationHelper.DetermineRelevantEmojis(emojiList, DM2UPI_SceneManager.Instance.settingsFile, DM2UPI_SceneManager.Instance.topEmojiPriority);

    DM2UPI_TranslatedPrediction translatedPrediction = new DM2UPI_TranslatedPrediction(newPredictedEmojiIDObjects.Count, inputText, jsonString,Guid.NewGuid().ToString(), Guid.NewGuid().ToString());
    for (int i = 0; i < newPredictedEmojiIDObjects.Count; i++)
    {
        translatedPrediction.predictedEmojiIDs[i] = newPredictedEmojiIDObjects[i].ID;
    }
    translatedPrediction.predictedEmojiIDObjects = newPredictedEmojiIDObjects;
    translatedPrediction = DM2UPI_CalculationHelper.DetermineEmotionShare(translatedPrediction, DM2UPI_SceneManager.Instance.settingsFile);
    translatedPrediction.toBeSaved = saveBinary;

    return translatedPrediction;
}
```

Abb. 67: DM2UPI Data Translator Code, VisualStudio Screenshot

```
[System.Serializable]
28 references
public class DM2UPI_TranslatedPrediction
{
    public string inputText;
    internal bool toBeSaved;
    public string predictionID = "";
    public string objectID = "";
    public DateTime dateTime;
    internal string rawJsonResponse;
    public int topEmojiCount;
    public int[] predictedEmojiIDs;
    public List<DM2UPI_EmojiIDObject> predictedEmojiIDObjects;
    public List<DM2UPI_EmotionShare> predictedEmotionShare;

    /// <summary>
    /// An object that contains all useful information of the prediction. This includes the original text input, a unique per prediction ID,
    /// a unique per session (and object) ID, the dateTime of the prediction, the top predicted emojis by ID and the predicted emotions
    /// with their corresponding percentageShare.
    /// </summary>
    /// <param name="newTopEmojiCount">number of emojis considered for the emotion percentageShare calculation</param>
    /// <param name="newInputText">original text input, the prediction is based on</param>
    /// <param name="newRawJsonResponse">the json respond from the deep moji server</param>
    /// <param name="newPredictionID">unique per prediction identifier</param>
    1 reference
    public DM2UPI_TranslatedPrediction(int newTopEmojiCount, string newInputText, string newRawJsonResponse, string newPredictionID, string newObjectID)
    {
        predictionID = newPredictionID;
        objectID = newObjectID;
        inputText = newInputText;
        rawJsonResponse = newRawJsonResponse;
        topEmojiCount = newTopEmojiCount;
        predictedEmojiIDs = new int[newTopEmojiCount];
        predictedEmojiIDObjects = new List<DM2UPI_EmojiIDObject>();
        predictedEmotionShare = new List<DM2UPI_EmotionShare>();
    }
}
```

Abb. 68: DM2UPI Translated Prediction Code, VisualStudio Screenshot

UNITY UI ELEMENTS

UI Elements ist Unity's neues UI Framework, das sich zu Teilen noch in Entwicklung befindet. Für Editor Erweiterungen ist es aber schon funktionstüchtig und ein Umstieg vom vorherigen IMGUI System ist früher oder später notwendig. UI Elements ist ein retained mode UI Framework, dass im Gegensatz zu IMGUI mit einer festen Objekthierarchie arbeitet, deren Struktur sich nicht von einem Frame auf den anderen grundlegend ändert (vgl. Campeanu 2019). Die bessere Skalierbarkeit und Performanz des neuen UI Systems waren aber nicht die Hauptgründe, warum ich mich dazu entschieden habe UI Elements für die Gestaltung meiner API zu nutzen.

Rechts sind drei Screenshots aufgeführt, die ausschnitthaft den Code für den Custom-Editor des DM2UPI_SceneManagers zeigen. Neben einer C# File, die die Logik beinhaltet, sind dort auch eine UXML File, die für die Hierarchie der Objekte zuständig ist und eine USS File aufgeführt, welche das Styling übernimmt. UXML und USS sind Unity's eigene Versionen von XML und CSS. Hier liegt die beachtliche Stärke von UI Elements. C# Code-Logik ist klar vom visuellen Aufbau der Editor Elemente und von ihrem Styling getrennt. Nicht nur lässt sich dadurch eine effizientere Aufgabenteilung von Programmierarbeit und visueller Gestaltung vornehmen, der Code lässt sich auch besser lesen und pflegen. Änderungen in USS Files zwingen Unity nicht dazu neu zu kompilieren und sind entsprechend sofort nach dem Speichern sichtbar.

UI Elements wird darüber hinaus in Zukunft auch für Unity's Runtime UI genutzt werden können. Demnach ergibt sich auch der Vorteil, für diesen Wandel gut vorbereitet zu sein, wenn man sich mit der Funktionalität des Frameworks bereits vertraut gemacht hat.


```
[CustomEditor(typeof(DM2UPI_SceneManager))]
0 references
internal class DM2UPI_SceneManagerEditor : Editor
{
    VisualElement m_RootElement;
    VisualTreeAsset m_SceneManagerVisualTree;
    DM2UPI_SceneManager dM2UPI_SceneManager;

    0 references
    private void OnEnable()
    {
        m_RootElement = new VisualElement();
        m_SceneManagerVisualTree = Resources
            .Load<VisualTreeAsset>("DM2UPI_Layout_CustomEditor");
        m_RootElement.styleSheets.Add(Resources
            .Load<StyleSheet>("DM2UPI_StyleSheet_CustomEditor"));
        dM2UPI_SceneManager = (DM2UPI_SceneManager)target;
    }

    0 references
    public override VisualElement CreateInspectorGUI()
    {
        var root = m_RootElement;
        root.Clear();
        m_SceneManagerVisualTree.CloneTree(root);

        root.Query<Image>(className: "dm2upi-sm-header-image")
            .ForEach(SetupImages);
        root.Query<Button>(name: "load-save-file-input-button")
            .ForEach(SetupLoadButton);
        root.Query<Button>(name: "load-all-save-files-input-button")
            .ForEach(SetupLoadAllButton);
        root.Query<Button>(name: "unload-save-file-input-button")
            .ForEach(SetupUnloadButton);
        root.Query<Button>(name: "unload-all-save-files-button")
            .ForEach(SetupUnloadAllButton);
        root.Query<Button>(name: "delete-save-file-input-button")
            .ForEach(SetupDeleteButton);
        root.Query<Button>(name: "delete-all-save-files-button")
            .ForEach(SetupDeleteAllButton);
        root.Query<Button>(name: "debug-prediction-button")
            .ForEach(SetupDebugButton);

        return root;
    }

    1 reference
    private void SetupImages(Image image)
    {
        var iconPath = "Images/"+image.name;
        image.image = Resources.Load<Texture2D>(iconPath);
    }

    1 reference
    private void SetupLoadButton(Button button)
    {
        button.clickable.clicked += () => dM2UPI_SceneManager
        .LoadSavedPrediction(dM2UPI_SceneManager.loadSaveFileInput);
    }

    1 reference
    private void SetupLoadAllButton(Button button)
    {
        button.clickable.clicked += () => dM2UPI_SceneManager
        .LoadAllSavedPredictions();
    }

    1 reference
    private void SetupUnloadButton(Button button)
    {
```

Abb. 69: DM2UPI Scene Manager Editor C# Code, VisualStudio Screenshot

```
<UXML xmlns="UnityEngine.UIElements" xmlns:editor="UnityEditor.UIElements">

    <VisualElement class="dm2upi-sm-container">

        <VisualElement class="dm2upi-sm-header-image-container">
            <Image class="dm2upi-sm-header-image" name="header-sm"/>
        </VisualElement>


        <VisualElement name="row" class="container">
            <editor:PropertyField binding-path="settingsFile"
                                  name="settings-file-field"
                                  tooltip="The settings file as
                                  configured by the DM2UPI_SettingsManager"/>
        </VisualElement>

        <VisualElement name="row" class="container">
            <editor:PropertyField binding-path="topEmojiPriority"
                                  name="top-emoji-priority-field"
                                  tooltip="Should the top relevant
                                  emojis be selected by a max amount
                                  or by a proportional threshold of
                                  their predicted percentage"/>
        </VisualElement>

        <VisualElement class="dm2upi-sm-header-image-container">
            <Image class="dm2upi-sm-header-image"
                   name="event-management"/>
        </VisualElement>

        <VisualElement name="row" class="container">
            <editor:PropertyField binding-path="linkedInputField"
                                  name="linked-input-field"
                                  tooltip="This input field is
                                  used within the SceneManagers
                                  PredictEmotionsByInputfield() function."/>
        </VisualElement>

        <VisualElement name="row" class="container">
            <editor:PropertyField binding-path="NewPredictionAvailableEvent"
                                  name="new-prediction-available-event"
                                  tooltip="This event is fired, whenever
                                  a new prediction is available. Subscribe
                                  to it, to notify your responding scripts."/>
        </VisualElement>

        <VisualElement class="dm2upi-sm-header-image-container">
            <Image class="dm2upi-sm-header-image"
                   name="data-management"/>
        </VisualElement>

        <VisualElement name="row" class="toggle-container">
            <editor:PropertyField binding-path="loadAllSaveFilesOnStart"
                                  name="load-all-save-files-onstart-bool"
                                  tooltip="Enabling this will load all
                                  saved predictions into sessionPredictions
                                  when the scene is starting."/>
        </VisualElement>

        <VisualElement name="row" class="toggle-container">
            <editor:PropertyField binding-path="savePredictionsDefault"
                                  name="save-predictions-default-bool"
                                  tooltip="Enabling this will save every
                                  prediction at a persistent data path"/>
        </VisualElement>

        <VisualElement name="row" class="file-path-container">
            <editor:PropertyField binding-path="loadSaveFileInput"
```

Abb. 70: DM2UPI Scene Manager Editor UXML Code, VisualStudio Screenshot

```
.dm2upi-sm-header-image-container {
    flex-direction: row;
    justify-content: flex-start;
    background-color: #282828;
    flex-shrink: 0;
    flex-grow: 1;
    margin-bottom: 5px;
}

.dm2upi-sm-header-image {
    align-self: auto;
    max-height: 47.5px;
    max-width: 253px;
    flex-shrink: 1;
    flex-grow: 0;
}

.file-path-container {
    flex-direction: row;
    margin-bottom:5px;
}

.container {
    margin-bottom: 5px;
}

.file-path-input {
    flex-grow: 1;
    flex-shrink: 1;


}

.dm2upi-sm-button {
    width: 120px;
    min-width:120px;
}

.toggle-container .unity-property-field
.unity-base-field .unity-toggle__input {
    justify-content: flex-end;
}

.file-path-input .unity-base-field .unity-label {
    min-width:0;
    padding-top:0px;
    padding-right:0px;
    width: 0px;
}
```

Abb. 71: DM2UPI Scene Manager Editor USS Code, VisualStudio Screenshot

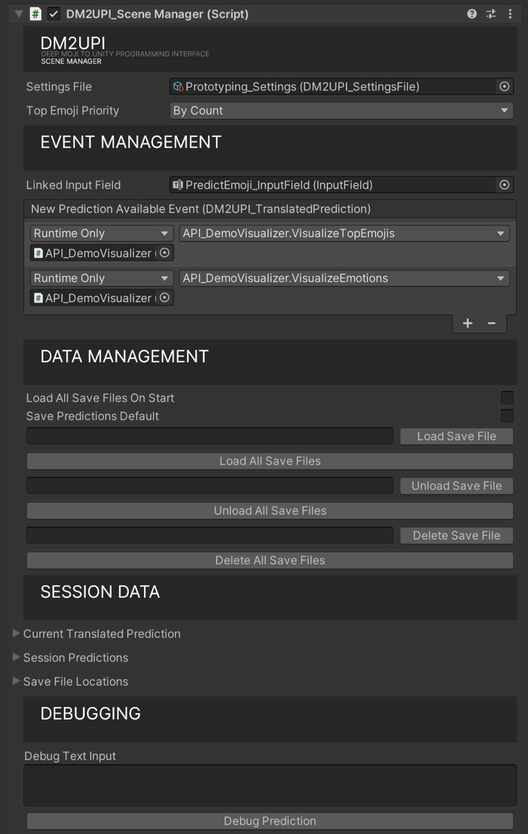


Abb. 72: Scene Manager Custom Inspector, Unity Screenshot

SCENE MANAGER CUSTOM INSPECTOR

Der Custom Inspektor für den Scene Manager, der in UI Elements erstellt wurde, soll möglichst viele Funktionalitäten der API auch über den Editor verfügbar machen. Dafür wurde durch das auf der letzten Seite zu sehende Styling eine zugängliche Listenansicht angelegt, die mit prägnanten Überschriften versehen wurde.

Das Settings File Objekt Feld lässt, wie von unityeigenen Objekten gewohnt, die direkte Auflistung und Auswahl aller, im Projekt befindlichen DM2UPI Settings Files zu. Die Zuweisung einer solchen Datei ist notwendig, da sich in ihr, wie bereits angeführt, alle Konfigurationsoptionen finden, die für die Berechnung der Vorhersagen genutzt werden. Das anschließende Dropdown Feld "Top Emoji Priority" lässt zwischen den zwei möglichen Einstellungen für die Reduzierung der relevanten Emojiobjekte wählen.

Unter Event Management ist zunächst ein Referenzfeld für ein Input Feld vorhanden, falls ein solches direkt für die Eingabetexte genutzt werden soll, anstatt diese via Code als String zu übergeben. Es schließt sich das "New Prediction Available Unity Event" an, dem auch hier im Inspektor neue Abonnenten hinzugefügt werden können.

Unter Data Management lässt sich zum einen festlegen, ob mit dem Szenenstart automatisch alle gespeicherten Saved Predictions geladen werden sollen, und zum anderen ob Vorhersagen als Binäre Dateien gespeichert werden sollen, wenn dies nicht spezifisch in die Predict Methode übergeben wurde. Es schließen sich mehrere Zeilen an, in die Speicherpfade von Saved Predictions eingegeben werden könne, um diese zu laden, entladen oder zu löschen.

Unter Session Data sind die letzte empfangene Vorhersage, alle Vorhersagen der Sitzung und die Dateipfade aller gespeicherten Vorhersagen aufgeführt.

Unter Debugging kann zu Testzwecken ein Text eingegeben werden, dessen Vorhersage anschließend in der Konsole geloggt wird.

Der Custom Inspector für den Scene Manager war mithilfe von UI Elements relativ einfach anzulegen, da keine Options- oder Layout verändernden Funktionalitäten vorhanden sind. Auf der vorherigen Seite ist zu sehen, dass hauptsächlich Buttons angelegt und mit bereits definierten Funktionen des Scene Managers verknüpft wurden. Darüber hinaus wurden Bilder als Überschriften eingebunden und die Anordnung und Abstände bestimmter visueller Elemente festgelegt. Die zugängliche Strukturierung via UXML hat diesen Prozess ebenso beschleunigt, wie die unmittelbare Anpassungsfähigkeit der USS Datei.

Die Programmierung des Settings Editor - Editor Windows war hingegen eine größere Herausforderung.

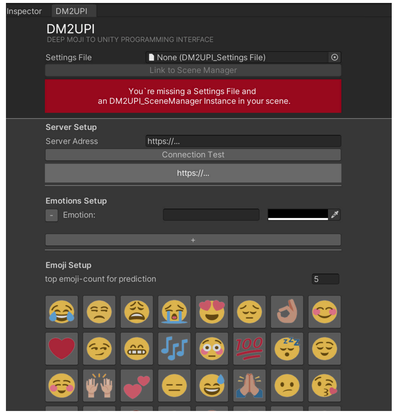


Abb. 73: Settings Editor 01, Unity Screenshot

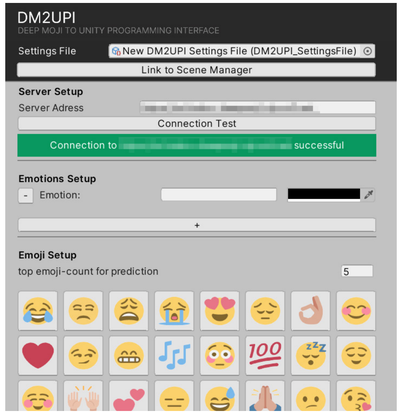


Abb. 74: Settings Editor 02, Unity Screenshot

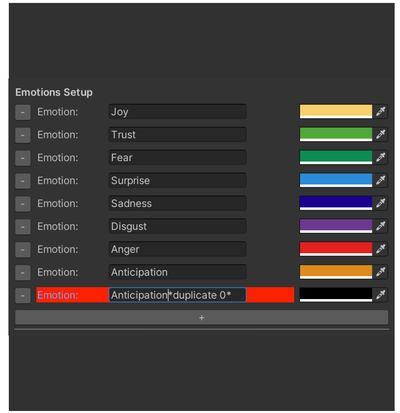


Abb. 75: Settings Editor Emotions Setup, Unity Screenshot

SETTINGS EDITOR WINDOW

Das Editor Window für den Settings Editor wurde in vier Hauptbereiche unterteilt. **Der erste Bereich** beinhaltet neben einem Header Bild und einem Objekt Feld, dass die zu konfigurierende Settings File auswählbar macht, auch einen Knopf der diese Settings File direkt mit dem Scene Manager in der aktuell geöffneten Szene verlinkt. Dadurch soll verhindert werden, dass möglicherweise eine Settings File konfiguriert wird, und im Scene Manager eine andere verlinkt bleibt. Die Optionen dieses ersten Bereichs bleiben immer am oberen Teil des Fensters fixiert, während sich die anderen Bereiche darunter scrollen lassen. Dies soll auch zur Übersichtlichkeit beitragen, da die aktuell bearbeitete Settings File immer visuell präsent bleibt. Dieser Bereich ist ebenfalls durch die Hintergrundfarbe von den anderen Bereichen abgegrenzt. Außerdem Teil dieses Bereiches ist ein Infofeld, das nur angezeigt wird, wenn für den Betrieb des DM2UPI benötigte Elemente nicht im korrekten Umfang bereitstehen. Dazu gehören: das Fehlen einer Settings File, das Fehlen eines Scene Managers oder auch das Vorhandensein von mehr als einem Scene Manager in der Szene.

Der zweite Bereich umfasst das Server Setup und stellt neben einem Eingabefeld für den Web-Link zum verwendeten Deep Moji Server auch einen "Connection Test" Knopf und ein weiteres Infofeld bereit. Wenn durch Drücken des Knopfes eine Testverbindung zum Server aufgebaut wird, wird das Ergebnis in diesem Infofeld angezeigt.

Der dritte Bereich ist das "Emotions Setup". hier können die Nutzer*innen nach Belieben eine unbegrenzte Anzahl Emotionen über den Plusknopf anlegen, eine Bezeichnung und eine Farbe zuweisen und über den Minusknopf wieder entfernen. Das System erkennt doppelte Eintragungen, markiert diese sichtbar für die Nutzer*innen, und benennt die Dopplungen direkt um, damit Emotionen immer einzigartig bleiben.

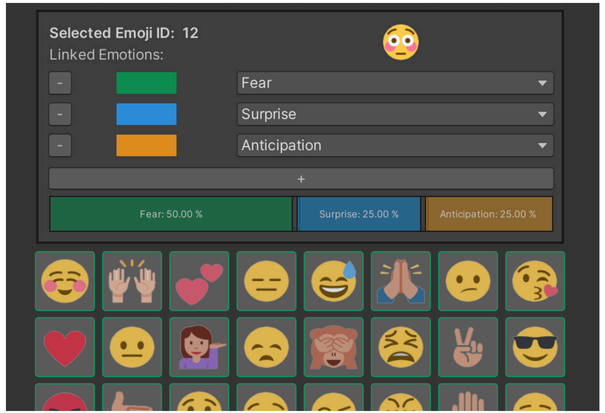


Abb. 76: Settings Editor Emoji Setup 01, Unity Screenshot

Abb. 77: Settings Editor Emoji Setup 02, Unity Screenshot

Diese Emotionen können im vierten Bereich den Emojiklassen des DeepMoji Modells zugeordnet werden. Zunächst lässt sich hier aber noch der Top Emoji Count und der top emoji threshold einstellen. An diese Eingabefelder schließen sich 64 Emoji Knöpfe an, die durch anklicken, in der Reihe jeweils unter dem ausgewählten Emoji, ein Konfigurationsfenster öffnen. In diesem Konfigurationsfenster lassen sich zuvor angelegte Emotionen durch den Plusknopf diesem Emoji zuordnen. Die Auswahl der Emotion erfolgt dabei über ein Dropdown Menü, das nach jeder hinzugefügten Emotion aktualisiert wird, und bereits vorhandene Emotionen aus dem Auswahlkatalog streicht. Es können dementsprechend auch nur maximal so viele Emotionen hinzugefügt werden, wie eingangs angelegt wurden. Bei Auswahl aller dieser Emotionen wird der Plusknopf anschließend ausgegraut und ist nicht mehr benutzbar. Unter den hinzugefügten Emotionen befindet sich ein Slider, der ein Segment pro Emotion beinhaltet. Durch das einfache Verschieben dieser Segmente lässt sich die prozentuale Gewichtung der einzelnen Emotionen in dem ausgewählten Emoji anschaulich und übersichtlich konfigurieren.

Unity nutzt solche Slider zwar beispielsweise in den LOD Settings der Engine, stellt aber keine Funktionalität über UI Elements bereit, die es ermöglicht einen solchen Slider für eigene User Interfaces zu nutzen. Nach ausgiebiger Recherche konnte ich einen Ansatz der Reddit-Nutzer*in Soraphis als Ausgangspunkt verwenden, um meine eigene Implementierung eines Multi-Value Sliders in das Editor Window mitaufzunehmen.

Die multiplen Abhängigkeiten, die durch die flexible Gestaltung dieses Settings Editors entstehen, waren eine große programmiertechnische Herausforderung, da jegliche Funktionalität selbst gecodet werden musste. Später einzustellende Werte (z. B. des Sliders) müssen immer auch berücksichtigen, dass nachträglich etwas an den Grundeinstellungen (z. B. den Emotionen) geändert werden kann. Wenn nach einer kompletten Konfiguration eine Emotion gelöscht, oder umbenannt wird, so müssen alle relevanten Werte in der Settings File überschrieben, Auswahlmöglichkeiten in den Dropdown Menüs neu sortiert, Prozentwerte aufgeteilt und das Editor Window neu gezeichnet werden.


```
975
                2 references
976             private void CreateMultipleValueSlider(int ID)
977             {
978                 List<float> tmpFloatList = new List<float>();
979
980                 int configuredEmotionsCount = settingsFile.emojiIDSetup[ID].emotions.Count;
981
982                 for (int i = 0; i < configuredEmotionsCount; i++)
983                 {
984                     tmpFloatList.Add(settingsFile.emojiIDSetup[ID].emotions[i].proportion);
985                 }
986
987                 emojiEmotionProportionPercentages = new float[configuredEmotionsCount];
988                 emojiEmotionProportionPercentages = tmpFloatList.ToArray();
989
990                 SetupMultipleValueSlider(emojiEmotionProportionPercentages);
991
992                 multipleValueSliderRoot = uxml_CurrentEmojiSetupContainer.Q<VisualElement>(className: "dm2upi-emoji-setup-percentage-slider-container");
993
994                 multipleValueSliderRoot.styleSheets.Add(Resources.Load<StyleSheet>("DM2UPI_StyleSheet_PercentageSlider"));
995
996                 multipleValueSliderRoot.Add(CreateMultiValueSliderContent());
997
998                 if (configuredEmotionsCount <= 1)
999                 {
1000                    multipleValueSliderRoot.visible = false;
1001                }
1002
1003
1004                multipleValueSliderUpdateAction = multipleValueSliderRoot.schedule.Execute(UpdateMultiValueSliderSizes);
1005                multipleValueSliderUpdateAction.Every(100); // ms
1006            }
1007
```

Abb. 78: DM2UPI Settings Editor Code 01, VisualStudio Screenshot

```
738             1 reference
739             private void AddEmotionDropdown(int ID)
740             {
741                 List<string> choicesWithoutActiveSelection = CalculateEmojiSetupDropdownChoices(ID);
742
743                 if (!settingsFile.emojiIDSetup[ID].emotions.Any(item => item.emotion.description == choicesWithoutActiveSelection[0]))
744                     settingsFile.emojiIDSetup[ID].emotions.Add(new EmotionBalance(settingsFile.emotionRange.Find
745                         (emotion => emotion.description == choicesWithoutActiveSelection[0]), settingsFile.emojiIDSetup[ID].emotions.Count == 0 ? 1f : 0.1f));
746
747                 DrawEmotionDropdown(ID, choicesWithoutActiveSelection[0], choicesWithoutActiveSelection);
748
749                 //if slider 0 add
750                 if (multipleValueSliderRoot == null || multipleValueSliderRoot.childCount == 0)
751                 {
752                     CreateMultipleValueSlider(ID);
753                 }
754                 else
755                 {
756                     //slider ++
757                     emojiProportions[emojiProportions.arraySize - 1] /= 2;
758                     emojiProportions.Insert(emojiProportions.arraySize, emojiProportions[emojiProportions.arraySize - 1]);
759                     multipleValueSliderRoot.RemoveAt(0);
760                     multipleValueSliderRoot.Insert(0, CreateMultiValueSliderContent());
761                     //Save to settings
762                     SaveProportionsToSettingsFile();
763                 }
764
765                 RedrawEmotionDropdowns(ID);
766                 EditorUtility.SetDirty(settingsFile);
767             }
```

Abb. 79: DM2UPI Settings Editor Code 02, VisualStudio Screenshot

```
1012        private void OnSettingsFileChanged(ChangeEvent<UnityEngine.Object> evt)
1013        {
1014            if (evt.previousValue != null)
1015                EditorUtility.SetDirty(evt.previousValue);
1016
1017            settingsFile = (DM2UPI_SettingsFile)evt.newValue;
1018
1019            if (settingsFile != null)
1020            {
1021                UpdateInputUXMLFields();
1022
1023                if (uxml_CurrentEmojiSetupContainer != null)
1024                {
1025                    uxml_CurrentEmojiSetupContainer.RemoveFromHierarchy();
1026                    uxml_CurrentEmojiSetupRows.Clear();
1027                }
1028
1029                previousEmojiID = -1;
1030            }
1031            else
1032            {
1033                if (uxml_CurrentEmojiSetupContainer != null)
1034                {
1035                    uxml_CurrentEmojiSetupContainer.RemoveFromHierarchy();
1036                    uxml_CurrentEmojiSetupRows.Clear();
1037                }
1038
1039                previousEmojiID = -1;
1040
1041                int count = uxml_EmotionConfigRowHolder.childCount - 1;
1042                for (int i = count; i >= 0; i--)
1043                {
1044                    uxml_EmotionConfigRowHolder.RemoveAt(i);
1045                }
1046            }
1047
1048            uxml_EmojiConfigButtons_Sorted.ForEach(LoadButtonOutlines);
1049            ValidateSceneView();
1050            ValidateLinkToSceneManagerButton();
1051        }


275         1 reference
276         private void SetupEmojiConfigButton(Button button)
277         {
278             // Reference to the VisualElement inside the button that serves
279             // as the button's icon.
280             VisualElement buttonIcon = button.Q(className: "dm2upi-emoji-config-button-icon");
281
282             // Icon's path in our project.
283             string iconPath = "Images/" + button.parent.name;
284
285             // Loads the actual asset from the above path.
286             Texture2D iconAsset = Resources.Load<Texture2D>(iconPath);
287
288             // Applies the above asset as a background image for the icon.
289             buttonIcon.style.backgroundImage = iconAsset;
290
291             button.clickable.clicked += () => EmojiButtonSelected(button, uxml_EmojiConfigButtons_Sorted.IndexOf(button));
292
293             // Sets a basic tooltip to the button itself.
294             button.tooltip = "Emoji ID: " + button.parent.name;
295         }
296
297         #endregion
298
299         #region Reactive UI Methods


1 reference
private void AddEmotionConfigRow()
{
    //get last index
    int lastIndex = uxml_EmotionConfigRowHolder.childCount;

    //add new emotion config row
    (Resources.Load<VisualTreeAsset>("DM2UPI_Template_EmotionConfigRow")).CloneTree(uxml_EmotionConfigRowHolder);

    //register callback to latest emotion config row TextField
    (uxml_EmotionConfigRowHolder[lastIndex].Q<TextField>(className: "dm2upi-emotion-config-row-input-field"))
        .RegisterValueChangedCallback((evt) => { settingsFile.emotions[lastIndex].name = evt.newValue; });
    //register callback to latest emotion config row Color
    (uxml_EmotionConfigRowHolder[lastIndex].Q<ColorField>(className: "dm2upi-emotion-config-row-color-field"))
        .RegisterValueChangedCallback((evt) => { settingsFile.emotions[lastIndex].color = evt.newValue; });
}

1 reference
private void LoadEmotionConfigRow()
{
    //delete all rows
    int count = uxml_EmotionConfigRowHolder.childCount - 1;
    for (int i = count; i >= 0; i--)
    {
        uxml_EmotionConfigRowHolder.RemoveAt(i);
    }

    //get count of all saved emotions
    int settingsCount = settingsFile.emotions.Count;

    //add all saved emotion config rows
    for (int i = 0; i < settingsCount; i++)
    {
        (Resources.Load<VisualTreeAsset>("DM2UPI_Template_EmotionConfigRow")).CloneTree(uxml_EmotionConfigRowHolder);

        //get references
        TextField textField = uxml_EmotionConfigRowHolder[i].Q<TextField>(className: "dm2upi-emotion-config-row-input-field");
        ColorField colorField = uxml_EmotionConfigRowHolder[i].Q<ColorField>(className: "dm2upi-emotion-config-row-color-fieldd");

        //set saved values
        textField.value = settingsFile.emotions[i].name;
        colorField.value = settingsFile.emotions[i].color;

        //buffer current index
        int currentIndex = i;

        //register callbacks
        textField.RegisterValueChangedCallback((evt) => { settingsFile.emotions[currentIndex].name = evt.newValue; });
        colorField.RegisterValueChangedCallback((evt) => { settingsFile.emotions[currentIndex].color = evt.newValue; });
    }
}
```

Abb. 80: DM2UPI Settings Editor Code Collage, VisualStudio Screenshots

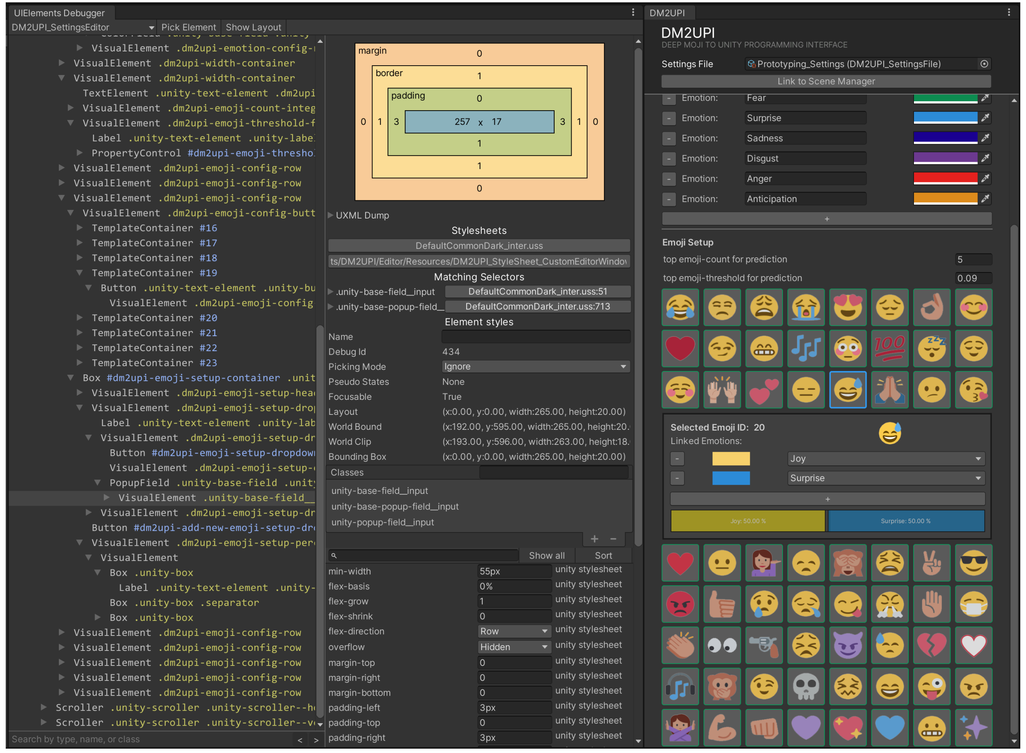


Abb. 83: UI Elements Debugger und DM2UPI Settings Editor Window, Unity Screenshot

UI ELEMENTS DEBUGGING

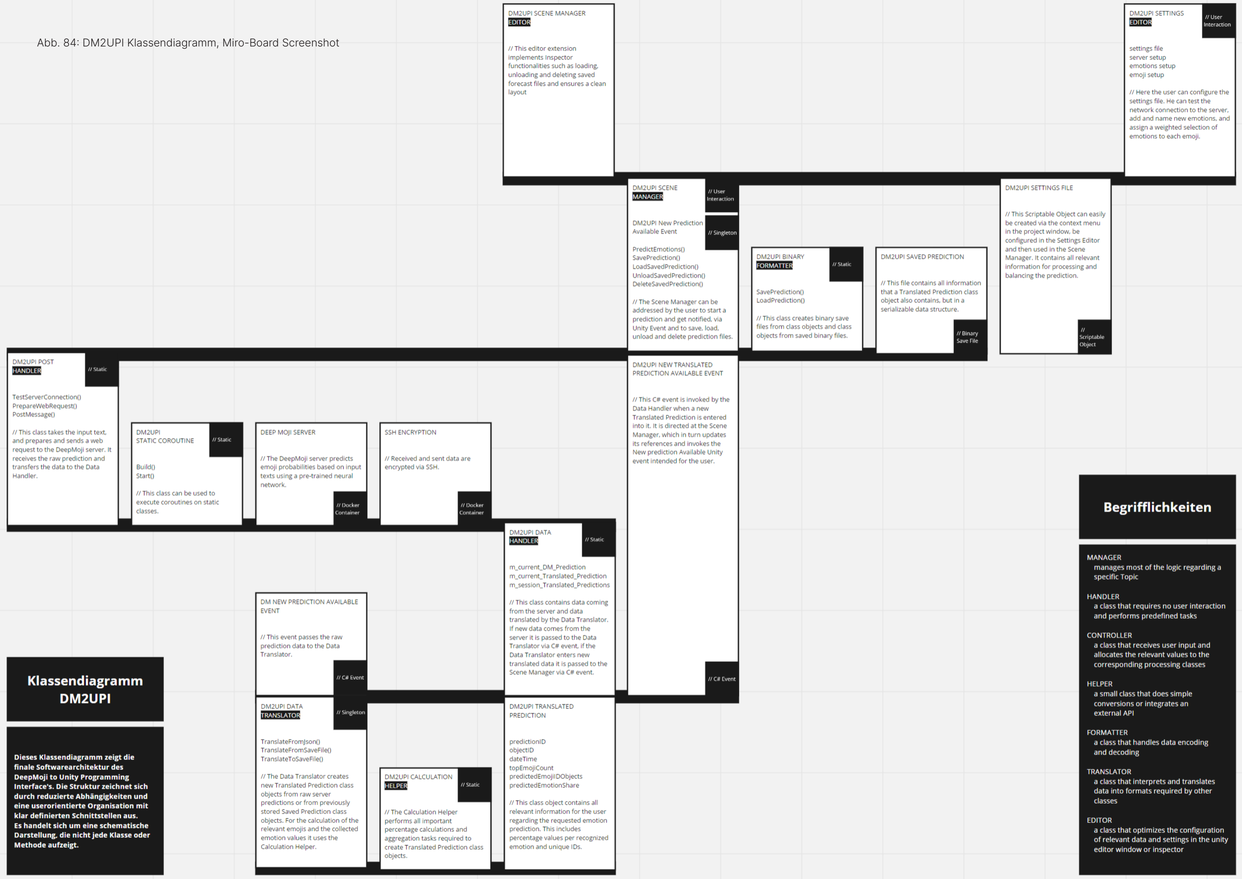
Unity UI Elements war zum Zeitpunkt dieser Arbeit noch nicht besonders umfangreich dokumentiert. Wenige Unite Talks auf YouTube, die lückenhafte online Dokumentation und vereinzelte Foreneinträge zu diesem Thema mussten genügen, um in das neue Framework einzusteigen.

Auch wenn die Nutzung nach dem Erlernen der Grundfunktionalitäten sehr intuitiv und verständlich ist, so kann der Einstieg doch etwas holprig sein, wenn man die Arbeit mit Unitys nun stückweise abgelösten Immediate Mode GUI gewöhnt ist.

Ein Tool, dass ich an dieser Stelle hervorheben möchte, da es mich einige Zeit gebraucht hat, bis ich eine Erwähnung durch meine Recherche gefunden habe, ist der UI Elements Debugger. Es lässt sich über das ‘drei Punkte’ Optionen Menü eines Unityfensters öffnen und bietet einem die Möglichkeit, einzelne Elemente eines anderen Unityfensters auszuwählen, hervorzuheben und zu inspizieren. Dazu bietet das Tool eine übersichtliche Baumstruktur, die den UXML Daten des ausgewählten Fensters nachempfunden ist. Somit wird die Hierarchie aller Fenster, die UI Elements bereits nutzen exemplarisch lesbar, was einem die Möglichkeit gibt an konkreten Beispielen zu lernen.

Darüber hinaus bietet der Debugger die Option USS Werte in Echtzeit zu testen, ohne den Code anpassen zu müssen, was sehr bei der Findung von Abstandswerten und der richtigen ‘flex’ Konfiguration hilft. Es ist also zu empfehlen, den UI Elements Debugger so früh wie möglich bei der Entwicklung von neuen UI Layouts zu benutzen.

Abb. 84: DM2UPI Klassendiagramm, Miro-Board Screenshot



CODE DOKUMENTATION

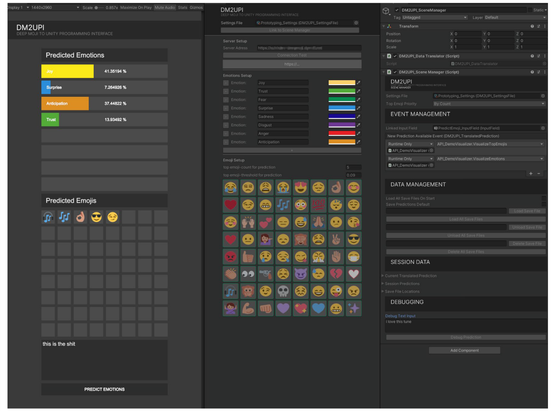
Um die Zugänglichkeit der API zu gewährleisten ist die Dokumentation des Codes und der Funktionalitäten in drei Hauptbereiche aufgeteilt. Zum einen sind user-adressierte Skripte mit Kommentaren versehen, die die Funktionsweise und die logische Abfolge der Methoden, sowie deren Abhängigkeiten untereinander erläutern. Darüber hinaus sind Funktionen und Klassen, die zur Verwendung durch die Nutzer*innen freigegeben sind, mit 'summaries' versehen, die für jeden 'overload' eine akkurate Beschreibung liefern, und die benötigten Parameter definieren.

Überdies ist dem DM2UPI Paket eine PDF-Dokumentation beigelegt, die im Folgenden aufgeführt ist. Diese Dokumentation enthält eine kurze Zusammenfassung des Projektursprungs, gefolgt von Referenzen auf das durch DM2UPI verwendete Deep Learning Model DeepMoji und die benötigte Serverumgebung, die beispielsweise durch Docker Container hergestellt werden kann. Anschließend wird beschrieben, wie der Settings File Configurator benutzt werden kann und welche Möglichkeiten durch das Editor-Interface gegeben sind. Im gleichen Umfang wird der 'custom inspector' des Scene Managers beschrieben.

Neben den Editor-spezifischen Möglichkeiten werden auch die 'public' Funktionen des Scene Managers aufgeführt und erläutert. Alle Beispiele sind mit aussagekräftigen Screenshots versehen, um die Nutzung so eindeutig wie möglich zu demonstrieren. Dass eine umfangreiche Dokumentation eines Softwareprodukts besonders dann wichtig ist, wenn die Zielgruppe aus Entwickler*innen besteht, ist eine Tatsache, die einem spätestens klar wird, wenn man selbst sein erstes unkommentiertes Plugin ohne Dokumentation verwendet. Dementsprechend sehe ich eine gründliche Dokumentation als wesentlichen Bestandteil dieses Designprojektes an.

Abb. 85: DM2UPI Documentation Seite 01

```
/// <summary>
/// DM2UPI – DOCUMENTATION
/// </summary>
/// <param name="DeepMoji to Unity Programming Interface">
```



// ABOUT

The DeepMoji to Unity Programming Interface is an easy to configure and accessible API for Unity developers based on a deep learning framework from MIT. The pre-trained model was built with millions of texts from Twitter containing emojis to generate a comprehensive database that can associate texts with specific emojis.

For more information about DeepMoji, please visit the dedicated website of MIT: https://deepmoji.mit.edu/about/

DM2UPI is a unity plugin that allows the individual configuration and evaluation of emotions in relation to the predicted emojis of a DeepMoji network. It includes a lightweight networking system, data management and event-based provision of structured classes with pre-calculated percentages and an integrated prediction file system. It also provides a comprehensive configurator with an efficient user interface structure that organizes the configuration of emotions and emojis and allows you to save different settings in separate save files.

// DEEP MOJI SETUP

To use this API, a connection to a DeepMoji server is required. The package includes a docker container optimized for the use on single-board-computers.

For more information on setting up and configuring a server using Docker, please refer to the extensive documentation on the Docker website:
https://docs.docker.com/

// SETTINGS FILE CONFIGURATION

The DM2UPI Settings File Configurator allows you to precisely set and save the evaluation parameters of the prediction. The configured settings file is then used by the "DM2UPI_SceneManager" to interpret your predictions. Since the file is a scriptable object, the handling, referencing and swapping is very simple and straightforward.

First look for a place in your project folder where you want to create your settings file in. Open the context menu with a right click and select "Create -> DM2UPI Settings File".

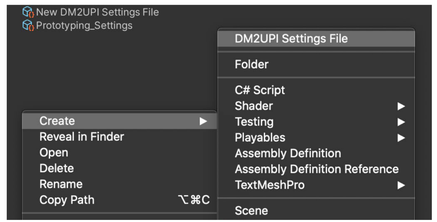
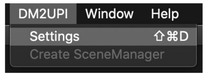
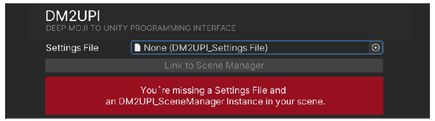


Abb. 86: DM2UPI Documentation Seite 02

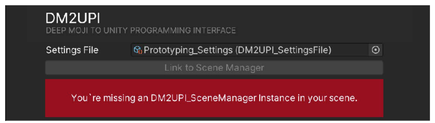
Then open the Settings File Configurator by opening the DM2UPI menu in the header bar and clicking on "Settings". Alternatively, you can use the shortcut **Ctrl + Shift + D** or **Command ⌘ + Shift + D** on Mac.



To use the DM2UPI Settings File Configurator please link the just created settings file in the first property field with the label "Settings File"



Furthermore, a "DM2UPI_SceneManager" is required in the Scene.



You can use the same header menu to spawn a scene manager in your scene. Then you can link the Settings File of the Settings File Configurator to the Scene Manager by clicking on the "Link to Scene Manager" button. That way you are making sure you'll be using the same file in your scene that you are about to configure.

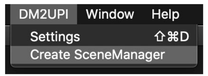


Abb. 87: DM2UPI Documentation Seite 03

// CONFIGURATOR OVERVIEW

Settings File
{
The Settings file, you are currently configuring.
}

Link to Scene Manager
{
This button links the settings file to the scene manager.
}

Server Address
{
The web address of your DeepMoji server.
}

Connection Test
{
this button tests the connection to the DeepMoji server and displays the result below.
}

Emotions Setup
{
Here you can create and edit the emotions DM2UPI should recognize.
}

Emotion
{
Consists of a unique description and a corresponding color.
}

Top Emoji-Count
{
The number of relevant emojis used for the prediction when the scene manager is set to "by count".
}

Top Emoji-Threshold
{
Lowest percentage of an emoji that will be included in the prediction if the scene manager is set to "by threshold".
}

Selected Emoji
{
Here you can assign emotions to the DeepMoji Emoji collection and adjust their proportion
}

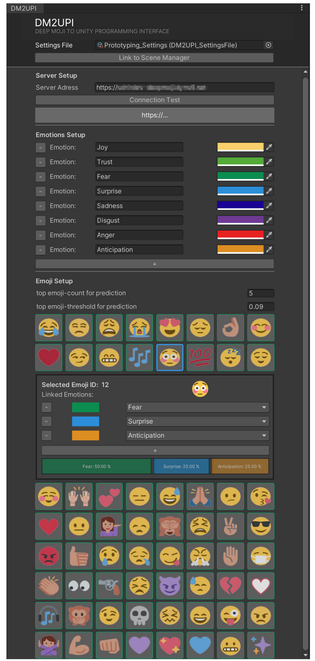
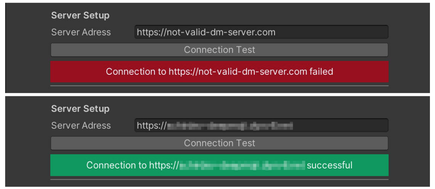


Abb. 88: DM2UPI Documentation Seite 04

// SERVER SETUP

Enter the web address where your DeepMoji server can be reached and do a short post test to see if DM2UPI could establish a connection.



// EMOTIONS SETUP

In the "Emotions Setup" you can add, edit and remove emotions. An emotion consists of a description - the name of the emotion - and a corresponding color, which you can use for visualization purposes.

Emotions that have already been set up and assigned to emojis can be renamed or removed without any problems. If you edit an emotion or its name, the change will be applied directly to all emojis you have previously configured. If you remove an emotion, all assignments of this emotion to the corresponding emojis will be removed and their proportional share will be split up among the remaining emotions. There is no upper limit for the number of emotions.

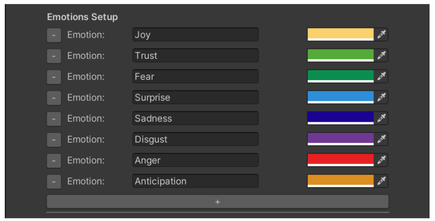
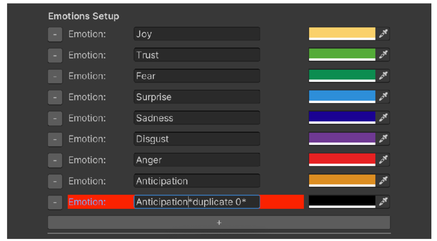


Abb. 89: DM2UPI Documentation Seite 05

Emotions must not be duplicated, since the evaluation of the predictions depends on a precise allocation of the percentage values. In case of a duplication, it will be noticeably marked and registered as a duplicate in the settings file.

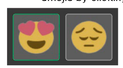


In order to set up a precise and complete configuration it is recommended to adapt an existing emotion model like the one of Robert Plutchik.

// EMOJI SETUP

The Emoji Setup is based on the 64 different Emojis used to train the DeepMoji algorithm. The DeepMoji server calculates the probability of each emoji corresponding to the text input. DM2UPI follows this calculation and categorizes the results according to the emotions you have defined.

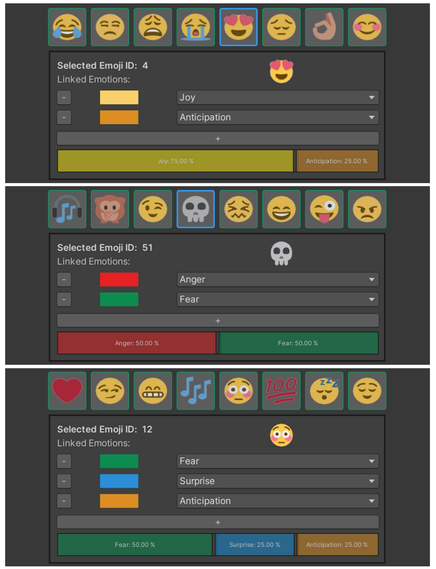
At this point you can now assign your previously created emotions to individual emojis by clicking on the emoji and opening the configuration window.



Already configured emojis are marked with a green outline to give you more guidance during the configuration process.

Each emoji can be associated with as many emotions as you have previously created. All assigned emotions share the percentage of each prediction that is assigned to the emoji. The slider in the configuration window can be used to balance the weight of the emotions.

Abb. 90: DM2UPI Documentation Seite 06



To ensure reliable results, it is recommended to use a representative survey that focuses on which emojis are usually used by your target group to communicate certain emotions. Keep in mind that emojis could be used differently depending on the demographic group.

Abb. 91: DM2UPI Documentation Seite 07

// SCENE MANAGER OVERVIEW

Settings File
{
The settings file, that is used for the calculation.
}

Top Emoji Priority
{
This setting determines whether a maximum number of emojis is included in the calculation or all emojis with a minimum percentage.
}

Linked Input Field
{
An input field that can be used for the prediction via PredictEmotionsByInputfield().
}

New Prediction Available Event
{
A Unity event that transmits the latest prediction.
}

Load all Save Files On Start
{
Loads all saved predictions when the scene is started.
}

Save Predictions Default
{
Determines whether predictions should be saved as binary files by default.
}

Load Save File
{
Enter the path of a file to load it, or load all files previously saved.
}

Unload Save File
{
Enter the path of a file to unload it, or unload all files previously saved.
}

Delete Save File
{
Enter the path of a file to delete it, or delete all files previously saved.
}

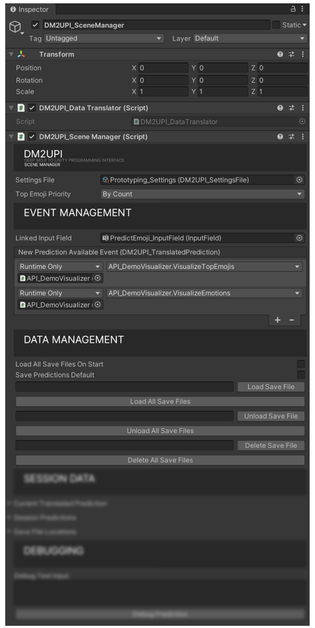


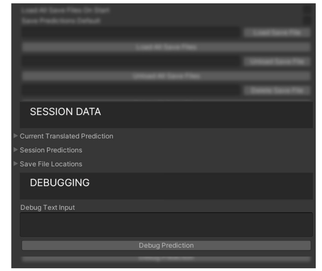
Abb. 92: DM2UPI Documentation Seite 08

Current Translated Prediction
{
The latest prediction.
}

Session Predictions
{
All predictions received and loaded in this session.
}

Save File Locations
{
All file paths of the save files of this project.
}

Debugging
{
With the "Debug Prediction" button you can perform an emotion check for the text you entered in "Debug Text Input". The result will be logged in the console.
}



// PREDICTIONS

To start a prediction, simply address the "DM2UPI_SceneManager" Singleton from your script. Please remember to include the DM2UPI namespace. Then use one of the following functions:

PredictEmotions()

PredictEmotions() starts the prediction based on a submitted string or the "debugTextInput". You can use the bool "saveBinary" to specify whether the prediction should be saved. If no bool is passed, the "savePredictionsDefault" value will be used.

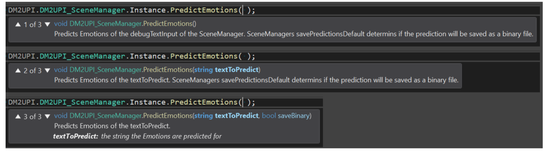
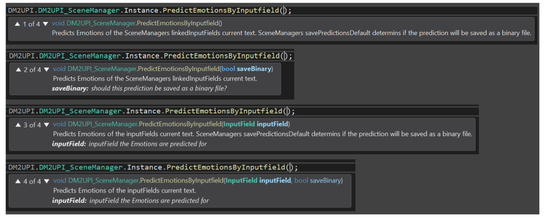


Abb. 93: DM2UPI Documentation Seite 09

PredictEmotionsByInputfield()

PredictEmotionsByInputfield() uses a passed input field or the input field linked in the scene manager for the prediction. You can use the bool "saveBinary" to specify whether the prediction should be saved. If no bool is passed, the "savePredictionsDefault" value will be used.



// EVENT SYSTEM

The "NewPredictionAvailableEvent" of the "DM2UPI_SceneManager" is always raised when a new prediction is calculated and registered in the scene manager. All subscribed functions will receive the latest "TranslatedPrediction".

The "NewPredictionAvailableEvent" is a Unity Event. You can assign your functions either via code or in the Inspector to receive the callback.

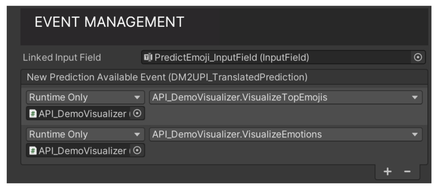


Abb. 94: DM2UPI Documentation Seite 10

```
DM2UPI.DM2UPI_SceneManager.Instance.NewPredictionAvailableEvent.AddListener(VisualizeTopEmojis);
DM2UPI.DM2UPI_SceneManager.Instance.NewPredictionAvailableEvent.AddListener(VisualizeEmotions);

DM2UPI.DM2UPI_SceneManager.Instance.NewPredictionAvailableEvent.RemoveListener(VisualizeTopEmojis);
DM2UPI.DM2UPI_SceneManager.Instance.NewPredictionAvailableEvent.RemoveListener(VisualizeEmotions);
```

// TRANSLATED PREDICTION

A "TranslatedPrediction" is transferred after each calculation via the "NewPredictionAvailableEvent" and added to the SessionPredictions. This class contains all the relevant information about the emotional content of the text you have just submitted.

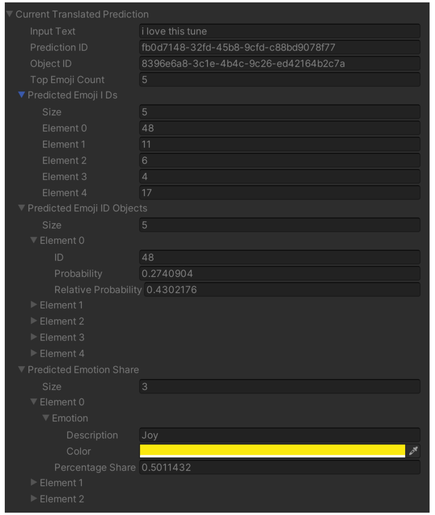


Abb. 95: DM2UPI Documentation Seite 11

Input Text
{
The text for which the emotions were calculated.
}

Prediction ID
{
A unique ID created specifically for this prediction, which is retained throughout saving, loading and scene change.
}

Object ID
{
A unique ID that applies to this translated prediction, but is newly generated when saving and loading.
}

Top Emoji Count
{
The number of emojis with the highest probability values included in the calculation. If the "Top Emoji Priority" in the scene manager is set to "By Count" this number will match the "Top Emoji-Count For Prediction" value in the settings file. If it is set to "By Threshold" the number indicates how many emojis have passed the threshold.
}

Predicted Emoji IDs
{
An array containing all the IDs of the emojis that have affected the calculation of the emotions.
}

Predicted Emoji ID Objects
{
A collection of all "EmojiIDObjects" that have affected the calculation of emotions. An "EmojiIDObject" consists of the ID of the emoji, the "Probability" which expresses the probability of this emoji in relation to all other emojis and the "Relative Probability" which expresses the probability of this emoji in relation to the other top emojis which were also relevant for the calculation.
}

Predicted Emotion Share
{
This list contains all the emotions recognized in the given text and their percentage share. The percentage share has been accumulated and weighted according to the configuration of all relevant emojis.
}

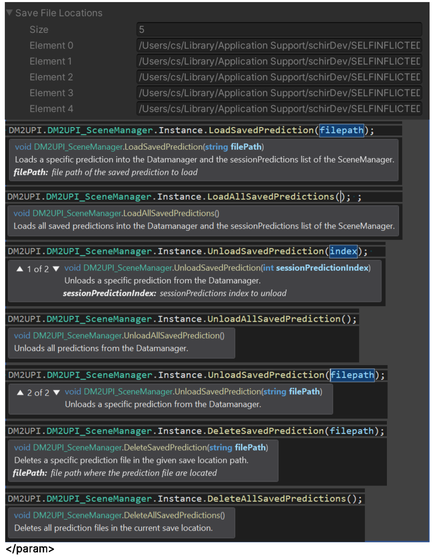
// SAVING, LOADING AND UNLOADING PREDICTIONS

Each prediction can be stored by the "DM2UPI_BinaryFormater". As storage location Unitys "Application.persistentDataPath" is used. Saved predictions are created as a ".sav" file. Additionally, a "FileNames.ref" file with references to the file names is created. The files can be loaded and unloaded in every session. A file can also be loaded multiple times if necessary.

Whether all saved files should be loaded for the scene start can be set in the scene manager. There you will also find a bool, which specifies the default save behavior. Every prediction can also provide the information if it should be saved or not inself.

Abb. 96: DM2UPI Documentation Seite 12

All current file paths can be read out in the scene manager. They are stored in the "Save File Locations" list. The file paths can be used in the inspector by the "DM2UPI_SceneManager" editor to load, unload and delete the save files. In addition, the scene manger contains public functions that you can control from your scripts to achieve the same effect.



</param>

Abb. 97: DM2UPI Documentation Seite 13

REFLEXION UND FAZIT

Zusammenfassend lässt sich über dieses Designprojekt in erster Linie sagen, dass mich die explorative Herangehensweise in viele unterschiedliche Entwicklungsbereiche geführt hat, deren Erkundung mich stets mit frischer Motivation für die nächsten Schritte ausgestattet hat. Ich habe in diesem Projekt viel gelernt und mich auch aus der Komfortzone der ausschließlichen Unity Entwicklung hinausbewegt. Mich mit dem selbstständigen Aufsetzen eines Server Client Systems mit selbst gehostetem DeepLearning Modell zu befassen, hat mir nicht nur die Bereiche Web-Deployment, Container-Management, Portfreigabe, Reverse-Proxy-Konfigurierung und Einplatinencomputersetup näher gebracht, es hat mir auch als Entwickler mehr Selbstvertrauen gegeben und mich bestärkt auch weiterhin Projekte in mir noch unbekannten Entwicklungsumgebungen umzusetzen.

Dieses Gefühl geht damit einher, dass ich mich bestätigt fühle weiterhin innovative Technologien zu nutzen, um deren Vermögen für interaktionsbasierte Medieninhalte im Allgemeinen und Videospiele im Besonderen zu untersuchen. Ich konnte die maschinelle Interpretation von Emotionen in meinen Playtests als nahezu eigenständiges spielerisches Element ausmachen, dessen Einbindung in Spielmechaniken, Kommunikationsprozesse und übergeordnete Gameloops zahlreiche, unterschiedliche Aufgaben übernehmen kann. Wichtig ist es, sich über die Wirkung in verschiedenen Kontexten bewusst zu werden, um auf der einen Seite den Designprozess so präzise wie möglich gestalten zu können und auf der anderen Seite Klarheit über moralische und inhaltliche Konnotationen zu erlangen, die ihre Spur in dem systemischen Geflecht gesellschaftlich-medialer Abhängigkeiten hinterlassen. Das Prototyping hat vor allem als Ideenfindungsprozess gut funktioniert. Viele Ansätze, die auch in diesem Projektbuch genannt wurden, konnten im Rahmen des Designprojekts aufgrund ihres Umfangs nicht umgesetzt werden. Um so mehr ermutigt mich die, als Werkstück hervorgegangene, API, da sie Folgeprojekte in Zukunft ohne großen Aufwand möglich machen wird.

Dass die Zielsetzung des Designprojekts während der Bearbeitungszeit zweimal maßgeblich überarbeitet wurde, war zudem eine Folge von Entscheidungsaufschub und einer überwältigenden Anzahl von möglichen Richtungen, die das Projekt nehmen konnte. So sehr ich meine Motivation der ergebnisoffenen Erkundung einer Thematik einerseits, und den kleinteiligen, akribischen Problemlösungsprozessen andererseits entnehme, so deutlich hat sich in dieser Arbeit für mich auch gezeigt, dass ich mir klar strukturierte Projektergebnisse definieren muss, um effizient arbeiten zu können. Die Beratungsgespräche mit meinem Erst-Prüfer haben in diesem Findungsprozess entscheidende Weichen gestellt.

Die Entscheidung, die Ergebnisse des Prototypings in die Entwicklung einer API einfließen zu lassen, hat viele offene Fragen geklärt und die Bedarfe durch die mir vertraute Zielgruppe eindeutig definiert. Der Anspruch an die Codestruktur, Zugriffsebenen, Code-Dokumentation und entkoppelte, nachhaltige und performante Softwarearchitektur ist gegenüber einem prototypischen Projekt grundverschieden, wenn man ein Produkt entwickelt, dass sich an Entwickler*innen richtet.

Auch an dieser Stelle konnte ich mich entscheidend weiterbilden und habe das Gefühl meine Programmierfähigkeiten nun auf einem breiteren Fundament zu wissen. Auch das Erlernen neuer Frameworks, wie Unitys UI Elements, ist für mich ein Zugewinn an Fähigkeiten, der sich aus dieser Arbeit ergibt und mir bereits Ideen für weitere Tools und Optimierungsmöglichkeiten für mein nächstes Projekt in den Kopf gesetzt hat.

Abschließend kann ich sagen, dass das Projekt SELF-INFLICTED EMPATHY mich auf eigentümliche Weise an meinen Werdegang durch das Bachelorstudium erinnert. Angefangen und angetrieben von einem Interesse an technologisch-medial kommunikativen Besonderheiten, über die spielerischen Versuche im Bereich der Visualisierung, mit der anschließenden Besinnung auf meine, durch die Arbeit aufgedeckten, Stärken im Bereich der Programmierung inkl. kurzem gesellschaftspolitischem Exkurs. So scheint mir diese Arbeit ein gebührender Widerhall und Abschluss meines Studiums zu sein.

AUSBLICK

Wie bereits erwähnt habe ich vor das DeepMoji to Unity Programming Interface für andere Entwickler*innen zur Verfügung zu stellen. Dafür bietet sich beispielsweise ein öffentliches Git Repository und eine Veröffentlichung über die Website itch.io an. Neben den in diesem Projektbuch aufgeführten Ideen für Installationen, oder auch dem ursprünglichen Tagebuchansatz, bin ich vor allem gespannt darauf, welche Einbindungen und Anwendungsbereiche nachfolgenden Game Designer*innen vorschweben.

Eine Kommilitonin hat sich in ihrer Abschlussarbeit beispielsweise mit dynamischen Blendshapes beschäftigt, die Emotionen in fotorealistischen Gesichtern eindrucksvoll erfahrbar machen. Hier ist womöglich eine direkte Anbindung unserer Projektergebnisse denkbar, die durch meine API erkannten Emotionen in der Mimik ihrer Modelle zum Ausdruck bringt.

Gerne würde ich auch eine kleine Game Jam organisieren, die als Constraint die Nutzung von Emotional Understanding vorsieht, um so weitere Eindrücke und Ideen zu sammeln. Darüber hinaus stehen mittelfristig für mich auch wieder Kooperationen mit Museen und Theatern an, die jeweils aus ihrer eigenen charakteristischen Perspektive auf die Nutzung dieses Frameworks blicken dürften.

Ob in Eigen- oder Fremdprojekten, ich werde die Nutzung von Neural Networks auch in Zukunft weiter verfolgen und möchte mich überdies auch mit anderen Deep Learning Modellen, wie beispielsweise dem Texte produzierenden GPT-2 Modell von Open AI beschäftigen - immer unter dem Vorzeichen, wie diese Technologien unsere Interaktion mit digitalen Systemen formen und neu definieren.

DANKSAGUNGEN

Sich einer Sache in Gänze verschrieben zu haben, heißt, Leidenschaft, Intensität und Energie auch auf andere übertragen zu können. In diesem Sinne gilt zuvorderst mein Dank Professor Thomas Bremer und Professorin Susanne Brandhorst für die unglaublich wertvollen Konsultationen, die Beratungsgespräche, Sprechstunden und die kreative Gestaltung der gesamten Studienzeit. Das zugewandte Feedback von Professor Bremer hat mir während dieser Arbeit wiederholt neue Wege aufgezeigt und mir geholfen meine Ideen ertragreich zu strukturieren. Ich danke euch beiden für ein außerordentlich lehrreiches und spannendes Studium. Auch allen Dozierenden des Studiengangs Game Design und den Mitarbeiter*innen des DE:HIVE möchte ich für ihren Beitrag zu diesem einzigartigen Studienerlebnis danken.

Ich danke allen Kommiliton*innen und Bekannten, die es auf sich genommen haben, meine nahezu 50 Minuten lange Umfrage zum Thema Emotionen zu Emojis auszufüllen. Die Ergebnisse dieses Pretests haben maßgeblich zum Featurekatalog meines finalen Werkstücks beigetragen.

Weiter danke ich meinen guten Freund*innen Anna Schiffels und Moritz Steinbeck für die gewinnbringenden Gespräche über Emotionen im Spiel, für den gegenseitigen Halt und das stets ehrliche Feedback. Privat oder professionell – auf euch ist immer Verlass.

Meinem Vater, Lucas Seng, möchte ich für den erdenden Dialog und den moralischen Beistand danken, auch und vor allem für das Bereitstellen seiner Wohnung als Arbeitsplatz während des Lockdowns in Folge der Corona-Pandemie. Egal wie ausgelastet du warst, hattest du stets ein offenes Ohr für mich.

Ich danke meiner Mutter, Sabine Schirdewahn, für ihre unterstützenden Worte und ihr beständiges Interesse an meiner Arbeit – Danke fürs Zuhören, Nachvollziehen und Kraft-Spenden. Ebenso danke ich Matthias Wagner K für zahlreiche geistreiche Gespräche und anregende Perspektivwechsel in Bezug auf die dieser Arbeit zugrunde liegende Thematik. „Viele sind hartnäckig in Bezug auf den einmal eingeschlagenen Weg, wenige in Bezug auf das Ziel."

Meinem alten Freund Friedemann Körner gilt mein Dank für seine Ratschläge in Sachen Datensicherheit und Serveradministration, meinem Mitbewohner Felix Mertineit danke ich für die regelmäßigen Unterhaltungen über neuronale Netzwerke während unserer Kaffeepausen. Ich freue mich schon darauf, diesen Austausch in Bezug auf weitere Deep Learning Projekte fortzuführen.

Ich danke meinen Freund*innen Yannick Pawils und Marie Günther für zahlreiche anregende Gespräche während des Studiums und darüber hinaus. Danke für eure motivierenden Musik-Playlisten, die mich über viele Arbeitstage und -nächte begleitet haben.

Für sein einzigartiges Gemüt und seine Fähigkeit, mich in stressigen Zeiten wieder auf den Boden zurückzuholen, danke ich meinem Freund Vincent Freyer. Es hat wohl einen tieferen Grund, dass wir nahezu jedes Projekt in diesem Studium gemeinsam angegangen haben.

Ich danke meinem guten Freund Samuel Auer, der mir den entscheidenden Anstoß gab, mich auf dieses Studium zu bewerben. Gleichermaßen danke ich Diemut Schilling, die mir im Vorfeld großzügig ihre Werkstatt zur Verfügung gestellt und mich für die Bewerbung gecoacht hat. Dank an Giuseppe Casciani, der – ebenfalls zu dieser Zeit – mit jedem Espresso, den wir gemeinsam zu uns nahmen, einen guten Ratschlag für mich parat hatte.

Abschließend möchte ich erneut meinen Eltern und meiner Familie für die unentbehrliche Unterstützung während meiner gesamten Studienzeit und das grenzenlose Verständnis für mein regelmäßiges Abtauchen während fordernder Projektphasen danken. In diesen Dank einschließen möchte ich auch meine Geschwister Carla und Timon, die meinem Leben stets eine spielerische Note geben und mich unablässig mit Bestimmung und Inspiration beschenken.

Und nicht zuletzt danke ich allen Freunden und Bekannten, die hier namentlich nicht genannt wurden, und die mir doch in diesem wichtigen Lebensabschnitt große Unterstützung haben zukommen lassen. Vielen Dank Ihnen und euch allen.

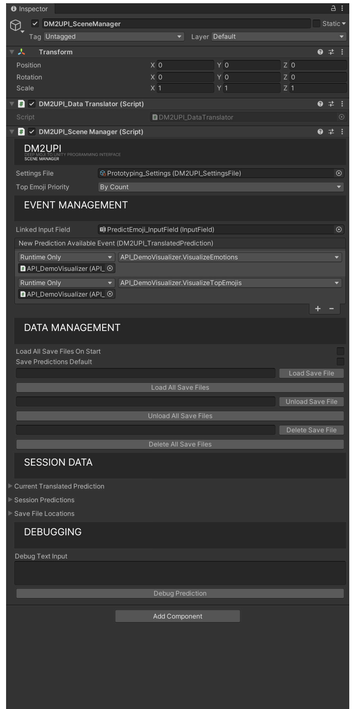


Abb. 98: DM2UPI Scene Manager

Abb. 99: DM2UPI Settings Editor

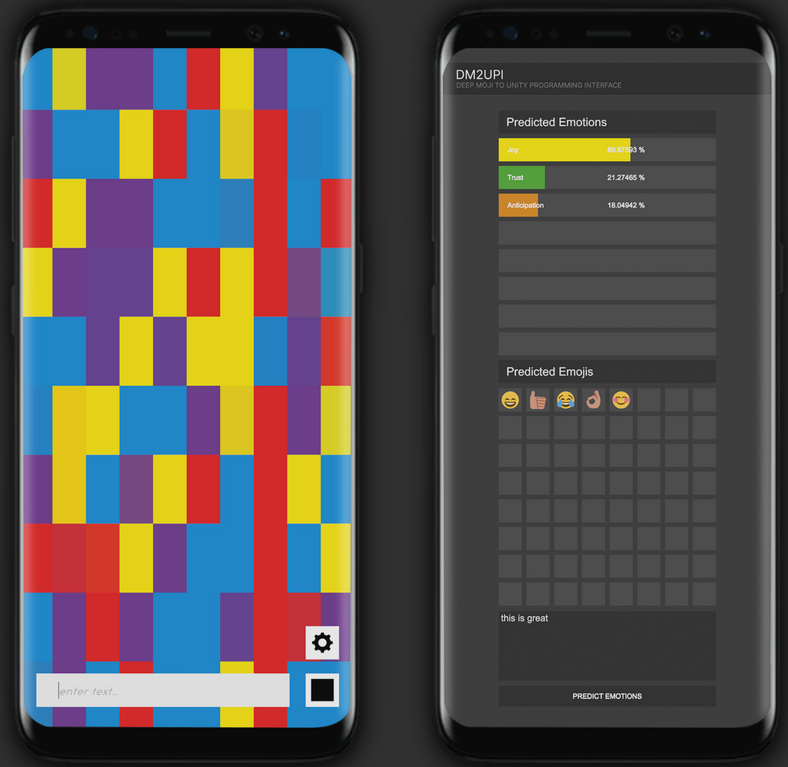


Abb. 100: Farben Prototyp und DM2UPI Smarphone Mockups, Android Screenshots

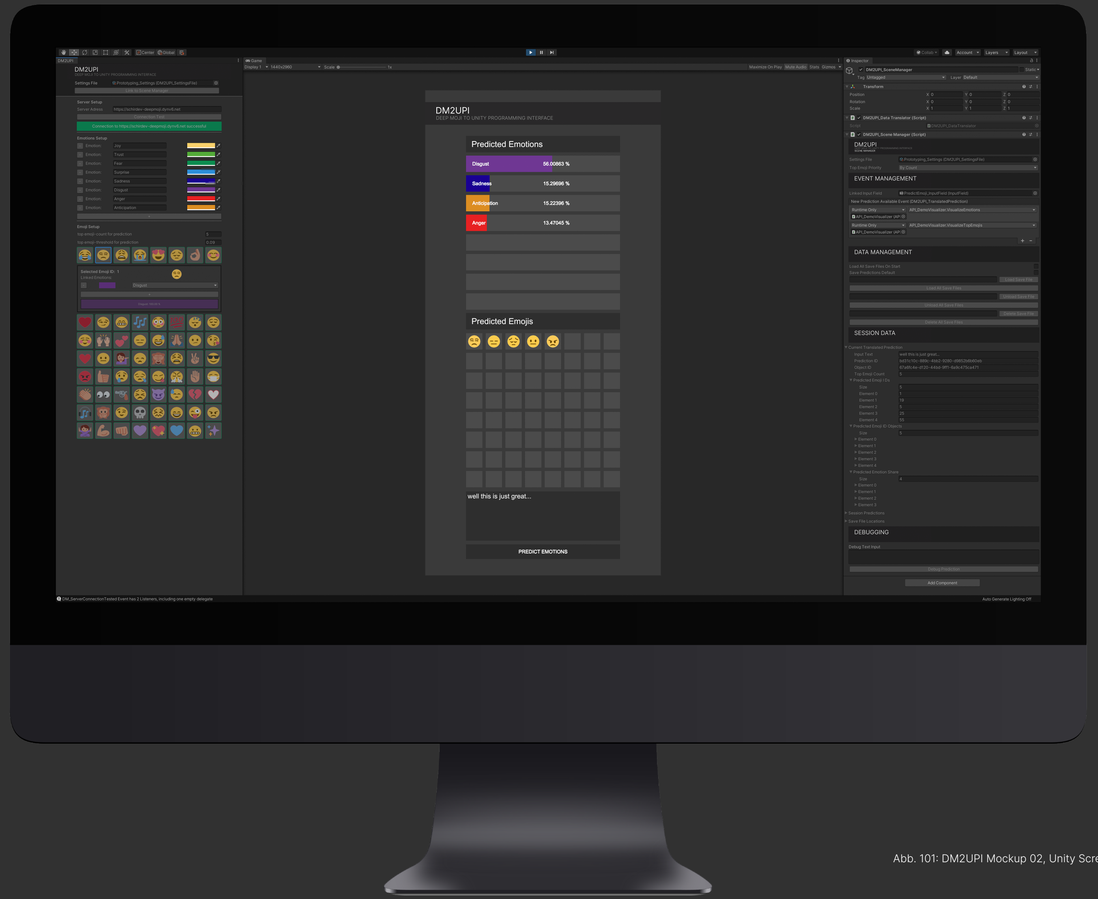


Abb. 101: DM2UPI Mockup 02, Unity Screenshot

ABBILDUNGSVERZEICHNIS

LITERATURVERZEICHNIS

- Berges, V.; Cohen, A.; Elion, C.; Gao, Y.; Goy, C.; Harper, J.; Henry, H.; Juliani, A.; Lange, D ; Mattar, M.; Teng, E.: (2020). Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627. https://github.com/Unity-Technologies/ml-agents.

- Campeanu, Damian (2019): What's new with UIElements in 2019.1, in: Unity Blog, URL: https://blogs.unity3d.com/2019/04/23/whats-new-with-uielements-in-2019-1/, 28.06.2020

- Felbo, Bjarke; Lehmann, Sune; Mislove, Alan; Rahwan, Iyad; Søgaard, Anders: (2017): Using millions of emoji occurrences to learn any-domain representationsfor detecting sentiment, emotion and sarcasm, in: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, URL: https://arxiv.org/pdf/1708.00524.pdf, 28.06.2020

- Gertz, Nolen (2018): Nihilism and technology, Rowman & Littlefield International, Ltd., London

- IBM o.V. (2019): Was ist Watson Tone Analyzer?, in: ibm.com, URL: https://www.ibm.com/de-de/cloud/watson-tone-analyzer?mhsrc=ibmsearch_p&mhq=Watson, 22.06.2020

- Kobbert, Max J. (2019): Das Buch der Farben, 2., ergänzte Auflage Jubiläumsausgabe, wbg (Wissenschaftliche Buchgesellschaft), Darmstadt

- Uros Krcadinac, Philippe Pasquier (2020): Synesketch, in http://metacreation.net, URL: http://metacreation.net/visual-arts-2/, 28.06.2020

- Wikipedia o.V (2020): Robert Plutchik, in: en.wikipedia.org, URL: https://en.wikipedia.org/wiki/Robert_Plutchik,28.06.2020

VERWENDETE SOFTWARE UND HILFSMITTEL

Für das Designprojekt und die Bachelorarbeit habe ich folgende Software, Hardware und Hilfsmittel verwendet:

Software

- Unity Game Engine 2019.2.11f1
- Unity Game Engine 2019.3.3f1
- Microsoft Visual Studio Community 2019 V. 16.3.9
- Microsoft Visual Studio for Mac Community V. 8.6
- Microsoft Visual Studio Code V. 1.46.1
- Ubuntu Terminal
- Windows PowerShell
- Cygwin Terminal
- Oh My Zsh
- Docker for Windows
- Docker Engine
- Adobe Creative Cloud Indesign V.15.1.1
- Adobe Creative Cloud Photoshop V. 21.2
- Adobe Creative Cloud Acrobat DC V. 20.0
- Microsoft Office Professional Plus 2019

Entwicklungsumgebung

- PC Arbeitsstation mit Windows 10
- Macbook Pro mit OSX
- Raspberry Pi 2B mit Raspberry Pi OS
- Intel NUC mit Windows 10
- LattePanda mit Ubuntu 16.04 LTS

Weitere Hilfsmittel

Zur Ideation und Dokumentation:

- miro.com/

Zur Versionierung:

- github.com/

Unity Plugins zum Prototyping:

- DOTween Pro: http://dotween.demigiant.com/
- TextMesh Pro: https://docs.unity3d.com/Manual/com.unity.textmeshpro.html
- Fingers Gestures: https://assetstore.unity.com/packages/tools/input-management/fingers-touch-gestures-for-unity-41076
- JSON .NET for Unity: https://assetstore.unity.com/packages/tools/input-management/json-net-for-unity-11347
- UCLA Wireframe Shader https://assetstore.unity.com/packages/vfx/shaders/directx-11/ucla-wireframe-shader-21897
- Realistic Star https://assetstore.unity.com/packages/3d/environments/sci-fi/realistic-star-39688

Für den selbst gehosteten Server:

- DeepMoji Modell: https://github.com/bfelbo/DeepMoji/archive/master.zip
- Let's Encrypt Docker https://hub.docker.com/r/linuxserver/letsencrypt/
- DynDNS Dienst: https://dynv6.com/
- Tensor Flow ohne AVX Instructions: https://github.com/maxhgerlach/tensorflow-1.8.0-ubuntu16.04-py27-no_avx-xeon_x5650/raw/master/tensorflow-1.8.0-cp27-cp27mu-linux_x86_64.whl

Tutorial zur Planetengenerierung:

- Tutorial von Sebastian Lague: https://youtu.be/QN39W020LqU

Grundlage für Multi-Value Slider:

- Demo von Git/ Reddit Nutzer*in soraphis: https://gist.github.com/soraphis/7c98d1bb77ac9cc088a629335b342a90

Grundlage für Static Coroutines:

- Code der Unity Forum Nutzer*in CykesDev: https://forum.unity.com/threads/c-coroutines-in-static-functions.134546/

Dokumentationen

- Unity Game Engine Documentation: https://docs.unity3d.com/2019.3/Documentation/Manual/index.html
- Docker Engine Documentation: https://docs.docker.com/
- Let's Encrypt Documentation: https://letsencrypt.org/docs/
- nginx Documentation https://nginx.org/en/docs/
- C# Documentation https://docs.microsoft.com/en-us/dotnet/csharp/

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und dabei keine anderen als die angegebenen Hilfsmittel benutzt habe. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach Publikationen oder Vorträgen anderer Autoren entnommen sind, habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Berlin, 20.04.2020

Caspar Schirdewahn

GAME DESIGN

DE:HIVE

Hochschule für Technik
und Wirtschaft Berlin
University of Applied Sciences