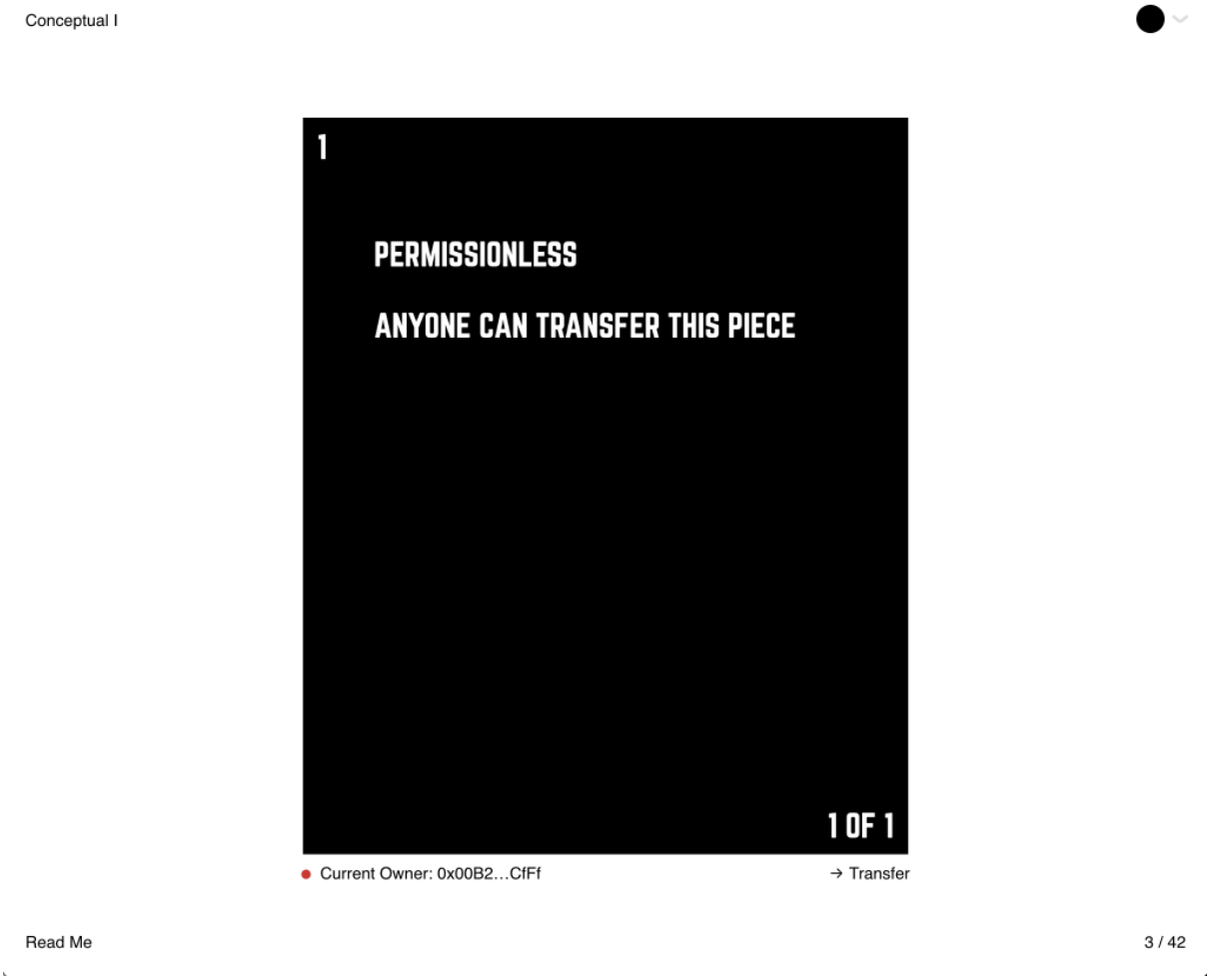
Dokumentation: Conceptual Series

David Simon, MTR / 15.01.2023

Die Arbeit «Conceptual Series» ist in Tandem mit dem Text «The Current Meta» entstanden und erprobt verschiedene technisch-medialen Möglichkeiten der Technologien, die NFTs unterliegen[1]. Sie ist ein Versuch, das Medium durch das methodische Ausprobieren weiter zu beleuchten und die Technologie *als Medium* zu erproben. Diese Auseinandersetzung birgt neben den Konzeptfragen auch mediale (und technische) Fragen: Ist das Konzeptkunst? Was *ist* das Medium? Ist der Code die Kunst? Was sind die richtigen formalen Entscheidungen zur visuellen Repräsentation eines solchen konzeptuellen Tokens? Vor allem wenn der Marktplatz zumeist der primäre Modus der Betrachtung ist[2].



Mit der Edition «Conceptual I» kann unter https://preview.conceptual.codes interagiert werden (Testnet Goerli). Der unterliegende *Smart Contract* kann unter https://contract.conceptual.codes betrachtet werden (ebenfalls in Teilen im Anhang dieses Dokuments). Den gesamten Quellcode inklusive des React Frontends und der Hardhat Tests stelle ich auf Anfrage gerne zur Verfügung

[1] Hier des Ethereum Protokolls, der EVM und des ERC721 Tokenstandards.
[2] Opensea, Gem, sudoswap etc.

3

OPENING HOURS

THIS PIECE HAS OPENING HOURS DURING WHICH IT EXISTS AND CAN BE TRANSFERRED

CURRENTLY CLOSED

Einordnung & Kontextualisierung

In «Conceptual Series» fliessen zwei Kunsthistorien zusammen. Zum einen die Konzeptkunst der 60er-Jahre (Joseph Kosuth) und ihrer Ausreiter und zum anderen die der *crypto art*, in welcher ebendiese konzeptuellen Ansätze teils bereits in den frühen 2010er Jahren probiert wurden. Beispielsweise Rhea Myers wegweisende Arbeit «Is Art» von 2014:

> *"Is Art" is an Ethereum contract that can be instructed to nominate itself as art (or not). Whoever toggles the contract's state as art sets it unimpeded until the next person sends a transaction to change it. A more rational system should be used - bidding, voting, a prediction market. The Duchampian aesthetic transubstantiation of artistic nomination is long played out. It is an art historical found object, as basic as a contract with a single bit of state. Brought together, the art historical and the contemporarily technological (or their audiences) can mutually animate and interrogate each other.*[3]

Ich zitiere im Folgenden relevanten Passagen meines Texts «The Current Meta», in welchen ein roter Faden der «crypto art» angerissen wird und teilweise auch gewisse innewohnende Logiken dieser Art von Kunst sich abzubilden beginnen:

> *Im Grunde genommen war für sie fast jedes Kunstwerk aus dieser aktuellen Welle eine Enttäuschung. Nur wenige Einzelne befragten dieses neue ökonomisch-mathematisch durchdrungene Medium wirklich selbst. Werke wie “Merge” von murat pak oder “Kudzu FoliaVirus” oder sogar Hirsts mit “The Currency”. Hier trat das eigentliche Medium zu Tage. Das Sarah Friends Foster Agreement zu ihren “Lifeforms” fand sie umso spannender, als dass sich darin das “Vertragliche” definitiv in einen analogen rechtlichen Vertrag verschob. Gerade die Lifeforms fand sie so pointiert gut, weil nicht-kodifizierte Verträge notwendig waren.*
>
> *In den frühen Phasen von crypto art waren die vermeintlich rechtlich-vertraglichen Aspekte ein immerwährender Fokus. Dieses initial konzeptuelle Missverständnis, das dem Namen smart contract entsprungen war, hatte als Projekt viele rechtliche und philosophische Versuche nach sich gezogen, diese autonomen, dummen scripts auch wirklich als Verträge zu lesen und per Erweiterung Code als rechtliche Schrift. Code is law in dieser Lesart war für sie eine naive Idee. In dieser Tradition verdiente, so fand sie, Rhea Myers eine besondere Würdigung. Ihre Werke wie “Is Art” spielten mit dem Vertrag als Kunstform. Auch Jonas Lunds neueste Arbeit “Smart Burn Contract” versuchte die Brücke zwischen traditionellem Vertrag und smart contract in einem neuen Versuch wieder einmal zu schliessen.*
>
> *Die Möglichkeit, Werte an programmatisch eingeschriebene Regeln zu koppeln, zu verwahren und zu transferieren, das war für sie zum Kern des*

[3] «Is Art, 2014/2015, Ethereum DApp», Rhea Myers, https://rhea.art/is-art

Mediums geworden. In dieser Fähigkeit, explizite Regeln einzuschreiben, lag in ihren Augen auch einer seiner spannendsten Aspekte. In der radikalen Transparenz und Irreversibilität entstand eine "reduzierte", strategische Umgebung gleichsam einem spieltheoretischen Gedankenexperiment. Jede Aktion war – wenn eingeschrieben – public knowledge und eine explizite Zustandsveränderung onchain.

[…]

Der Begriff crypto art fasste für sie eine Bewegung der frühen Bitcoin-Jahre bis hin zu den Anfängen von NFTs. Eine vielfältige Sammlung von Auseinandersetzungen mit einem kryptischen, kryptografisch-ermöglichten Phänomen. Die Wertfrage war in diesen Zeiten eine Behauptung mit wenig Resonanz. Werke wie das @coin_artist's & Myer's 1Flamen6 Gemälde "TORCHED H34R7S." waren Versuche, Werte in das Werk selbst einzubinden. In diesem Fall waren 5 Bitcoins, in den Mustern des Werks versteckt eingebettet. Vermutlich bedeutete die Verabschiedung des ERC721 Standards auch das Ende von crypto art. Die Spezifikation standardisierte und zementierte in einem gewissen Sinne die Kategorie NFTs. Im kollektiven Durchdeklinieren der Zehntausender-Formel von crypto punks in 2021 fühlte sie die gleiche Goldgräber- stimmung, die sie bereits 2017 empfunden hatte. War es wirklich nur die Suche nach dem Gold? Nur Reichtum? Kam hier nicht auch ein Ausdruck von Kultur und Kreativität zutage, die über eine einfache permutative Logik hinausging?

[…]

Damals hatte sie nicht begriffen, dass sie mit einer sich später immer mehr zeigenden Regel gebrochen hatten: Sammlungen mussten einzigartig und fungibel sein. Es gab einen Differenzierungszwang des einzelnen tokens, der sich aber nicht so weit erstrecken durfte, dass die Vergleichbarkeit zu stark litt. Editionen mussten zwar homogen sein und doch individuiert werden, ansonsten war das Einzelne als solches nicht mehr erkennbar.

Es gab eine Art Genre von Projekten, deren ästhetische Dimension sich bewusst kryptisch gab. Das Obskure war in den Mechanismen des spaces angelegt. Das Gold, das auf der Oberfläche lag, war manchmal zu offensichtlich: Ein Werk musste schwierig to unearth sein. Code und Codex. Verschleiert und verhüllt. Sie erinnerte sich an ein Buch, das sie letztens bei einer Visite in den Händen gehalten hatte mit der Aufschrift "Art is a Problem". Vielleicht musste Kunst problematisch sein? War sie sonst irrelevant? Sogar hier hatte art diese Rolle eingenommen, zu brechen, zu appropriieren und zu hinterfragen. Trotzdem war der grosse Teil der art nur geschmacklich problematisch. Spätestens mit sudoswap und den floor pools waren aus den NFT-Sammlungen letztlich wirklich nur altcoins with pictures geworden.

In der zweiten Hälfte des Texts münden die Gedanken dann im Angewandten. Hier tritt auch die Erkenntnis zu Tage, dass ich den Umgang mit der Fähigkeit, beliebige Regeln und Einschränkungen zu schaffen, als eine grossteils vernachlässigte Dimension empfinde. Diese Erkenntnis mündet dann letztlich in ersten Konzepten der vorliegenden Edition:

> *Vielleicht war es Zeit, einen weiteren Schritt zu tun? Sie beschloss, eine Serie zu beginnen, in welcher sie die Eigenschaften und dadurch das konzeptuelle Potential des Mediums methodisch durchexerzieren würde. […] Das Werk musste eine Edition von NFTs werden – ein anderes Medium wäre eine Ausflucht. Vieles dessen, was sie hier zu schaffen gedachte, war nicht neu. Mechanismen und Konfigurationen würde sie extrahieren und bereinigen von jeglichem verschleiernden Firlefanz. Natürlich waren davon einige untrennbar mit einer Metapher oder einem Konzept verbunden. Sie würde, entschied sie, diese Metaphern tolerieren. Vielleicht war sogar zumeist ein metaphorisches Verständnis Voraussetzung für ihre ästhetische Wahrnehmung von Code? Wäre das Werk von pak "Merge" ohne die Assoziation von physikalischer Masse und Grösse von gleicher ästhetischer Wirkung?*
>
> *Durchaus würden einige tokens dieser Edition eine trockene, rein mechanische Implementation einer strukturellen Gegebenheit bedeuten. Sie dachte zum Beispiel an ein token, das sie "Coinbase" nennen würde und das nur von der block.coinbase Adresse empfangen werden könnte. Das token würde also nur in den Besitz gelangen von Adressen, welche aktiv am grundlegenden Mechanismus partizipierten, welcher das Netzwerk antreibt und konstituiert. "Coinbase" fasste eine konzeptuell geschlossene Idee, deren Wirkung sich aus einer einzigen Restriktion und Regel ergab. Einschränkungen und Regeln, dies war im Grunde genommen die vernachlässigte Dimension dieses Mediums.*
>
> *[…]*
>
> *In dieser konzeptuellen Edition würden gewisse Setzungen unwirksam in Isolation sein. Ihre Wirkung zeigte sich erst in der Interaktion mit Anderen. Es würde also dann tokens geben, die beispielsweise nur in Abhängigkeit von anderen tokens transferiert oder erstattet werden könnten. Und radikaler noch: Ihre Transferierbarkeit als direkte Funktion eines Gegenübers formuliert. Beispielsweise eine Folge von 7 tokens, von welchen nur jeweils dasjenige transferierbar war, dessen besitzende Adresse die reichste war. Trotz der Radikalität ihres Ansatzes fühlt sie sich dazu verführt, rein konzeptuelle tokens zu erstellen, die funktional nicht bemerkenswert waren und deren Weisung einzig durch ihren eingebetteten Text erfolgte.*
>
> > *OFF-CHAIN*
> > *Ownership of this piece is granted by a paper certificate*
> > *signed by the artist. In all other respects it is unremarkable.*

In diesem Akt steckte die Frage nach einem Bruch mit der Prämisse. War das ein hack? Die Idee eines hacks hatte für sie eine Konnotation des Unvorhergesehenen: Ein Missbrauch und wie auch Schlupfloch. Bis heute beriefen sich attacker im space auf die Phrase code is law*. Mit einer Schwachstelle wurden Millionen gestohlen und moralisch so gerechtfertigt, dass der Code, der diese Werte sicherte, als so etwas wie ein Gesetz galt. Legal but not right. Full-circle, dachte sie. Sie war wieder am Kern der Sache angelangt. Regeln.*

Der Bruch mit Regeln – selbst ihre eigenen –, war ein dem Thema innewohnender Aspekt und sie entschied, dass sie dies noch weiter treiben müsste. Der Standard des ERC721 gab den ersten Rahmen vor, den sie sich entschied, zu brechen. Die Existenz eines gegebenen tokens nach diesem Standard liesse sich kontextabhängig so modifizieren, dass ein token nur zur Hälfte der Zeit existiert. Oder spezifische Öffnungszeiten hätte.

Eigentlich mochte sie clevere Kunst nicht. Kunst, die sie bewegte, empfand sie als "wahr". Unprätentiös, ehrlich, direkt und ästhetisch wahr; so würde auch diese Serie erscheinen müssen. Ein unerreichbarer Anspruch – ein Ideal. Es war unmöglich, ein Medium final abzuwickeln. Dieses Werk würde lückenhaft, unkomplett, ein Abdruck der Zeit sein.

NFTs waren eine Zeitlang quasi Synonym für eine generative Ästhetik. Auch wenn «Conceptual Series» selbst kein generatives Kunstwerk ist, enthält selbst diese sorgfältig kuratierte, handverlesene Sammlung auch wiederum generative Mechanismen, die als «Primitive» – also als Grundbausteine oder «Patterns» – der letzten Welle von Anfang 2021 entsprungen waren. Hier ein spezifischer Mechanismus, der einen Input deterministisch in einen Output wandelt: Von einer Ethereum-Adresse zu räumlichen Koordinaten in Nr. 21 «LOCUS»[4]:

Generative Kunst war eine Art Meta-Gestaltung. Es war im Kern das Gestalten eines Algorithmus – einer Rezeptur? Unter verschiedenen Inputs würde es im besten Fall immer einen interessanten Output produzieren. Das Spiel mit dieser Exprimierung und den emerging Strukturen und Werken kam ihr natürlich. Es war eine prozedurale Logik, deren Einzug sie schon eine Weile in allen Medien hatte sehen können. Die banalste Form des Generativen war das, was rarities genannt wurde. Ein Zusammenwürfeln von verschiedenen layers oder attributes die in einer Sammlung von besagter Semi-Differenziertheit. Die spannende Form des Generativen, fand sie, zeigte sich zum Beispiel in Tylers Fidenzas oder in den etwas esoterischen "Mutant Garden Seeders" oder auch in Jan Roberts "Ornaments" und natürlich die arkanen "Mathcastles". Bei solchen Werken empfand sie das Ergebnis effektiv als ein Exprimieren. Hier trat das Werk als ästhetischer Ausdruck der codierten Rezeptur zu Tage.

---

[4] «This piece will output three spatial coordinates for the current address. It can be used to spatially relate addresses».

Die geschaffene Edition an Tokens setzt ein Wissen über das Medium und zumindest in einfachen Zügen ein Verständnis der unterliegenden Technologien voraus. Beispielsweise existiert ein NFT im traditionellen Sinn[5] erst, nachdem es das erste Mal “transferiert” wird, sozusagen aus dem Nichts (0x0) auf eine Adresse. Darin sind gewisse ontologische Fragen angelegt, die in der Edition thematisiert werden. Hier beispielsweise indem die Existenz – genauer genommen die Auskunft über die Existenz – einiger Tokens auf verschiedene Weisen kontextabhängig modifiziert wird. Doch bleiben diese Ebenen nicht bloss Insidern zugänglich? Ist dieser «gap» temporär?

> *Ein wiederkehrendes Problem war der* gap*. Sie beherrschte diese* techne *– sie war eine stolze Artisane. Aber war das nicht die Voraussetzung, um diese Art der Ästhetik überhaupt schätzen zu können? War die technische “Eingebundenheit” die neue, zweite Geschmacksbildung? War diese Kunst eine hermetische, unzugängliche und letztlich elitäre Angelegenheit oder war es einfach früh? Selbst sie hatte manchmal diese Ängste, abgehängt zu werden. Die Rate der Neuerungen nur schon hier im space beschleunigte sich oft explosionsartig exponentiell. Und doch, die Prinzipien blieben dieselben, non?*

[5] Heisst hier: EVM basiert und dem ERC721-Standard folgend.

Technische Erläuterung: ERC721, Regeln und Mechanismen

NFTs basieren auf dem Token-Standard ERC721. Dieser spezifiziert, wie sich ein Smart Contract, der den Standard (und damit NFTs) implementiert, verhalten muss bzw. welche Informationen und Funktionalitäten er wie exponieren muss. Hierin – im Fakt, dass der Smart Contract selbst implementiert werden kann – liegt die Grundlage dafür, dass die Verhaltensweisen der einzelnen Tokens modifiziert, erweitert und umgenutzt werden können.[6]

Das Token Nr. 2 «NO EXTERNALITIES» kann nur von einem Smart Contract besessen werden. Das bedeutet das Token kann nicht auf eine konventionellen Ethereum Adresse (EOA)[7] transferiert werden. Diese Kondition lässt sich mit drei einfachen Zeilen beschreiben, welche sicherstellen, dass die bei einem Transfer empfangende Adresse Code enthält, was nur Smart Contracts können und keine EOA.

```
// This piece can only be held by a contract
uint size;
assembly { size := extcodesize(to) }
require(size > 0);
```

Mehrere Tokens in der Edition schränken auf ähnliche Weise ein, auf welche Adressen sie transferiert werden können. Es gibt mehrere Tokens, die bestimmte hexadezimale Zahlenfolgen am Anfang und Schluss einer Adresse benötigen. Beispielsweise Token Nr. 4 «DEAD BEEF».

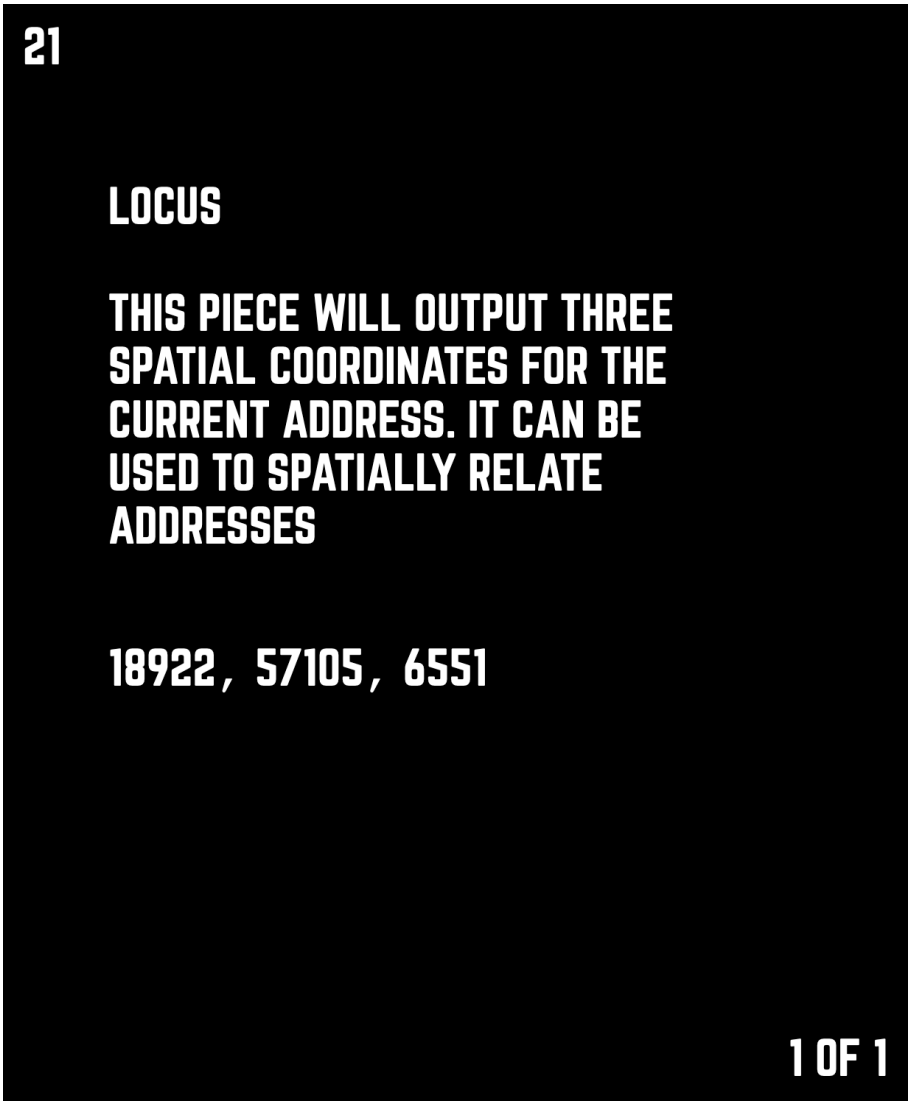
```
// This piece can only be held by an address starting
// with DEAD and ending with BEEF
bytes20 addr = bytes20(to);
if (
    !(addr & hex"ffff0000000000000000000000000000ffff"
          == hex"dead0000000000000000000000000000beef")
) {
          revert("No beef");
}
```

Diese Bedingungen und Regeln sind in einem sogenannten «Hook» platziert, der jedes mal aufgerufen wird, wenn das Token transferiert werden soll (auch beim aller ersten Transfer *ex nihilo).* Hier lassen sich also diverse Umgebungsvariablen überprüfen und einschränken, z.B. eben auch der intendierte Empfänger (im oberen Beispiel die Variable «to»).

[6] Der gesammte Haupt-Contract ist im Anhang zu diesem Dokument ersichtlich.
[7] Eine konventionelle bzw. Reguläre Adresse nennt sich eine «Externally Owned Address».

Technische Erläuterung: Onchain-Rendering und Selbstsuffizienz

Eine technische Besonderheit, die in *crypto*-Kreisen besonders wertgeschätzt wird und sich teilweise zu einer Art eigenem Genre entwickelt hat, ist Kunst, die «onchain»[8] gerenderet wird. Das bedeutet so viel wie, dass der Smart Contract selbst das Bild generiert und keine externe Hoster oder Protokolle notwendig sind, wo eine etwaige Bilddatei abgelegt werden müsste. Diese Edition ist in diesem Sinne komplett selbstsuffizient, da die Bilder direkt «onchain» gerenderet werden, und keine Abhängigkeiten von Dritten oder mir als Künstler bestehen. Die Schrift und das Layout ist direkt eingebettet in den Smart Contract und kann abgerufen werden, um ein SVG Bild zurückzuerhalten. Dies macht auch dynamische Tokens möglich, wie Nr. 27 «1 – *Anyone can increase the title of this piece».* In diesem Token eingebettet als Titel ist eine Zahl im Contract, welche von jeder beliebigen Adresse in einer Transaktion um +1 erhöht werden kann.



[8] «Onchain» als Schreibweise analog zu «online». Oft wird auch «on-chain» verwendet.

Formale Aspekte & Gestaltung

Die radikalste Form der Umsetzung wäre eine komplett abstrakte. Allerdings gilt es bei NFTs ihre primäre Betrachtungs- und Rezeptionsweise zu bedenken, die meist direkt im Marktplatz selbst geschieht:

> *Sie setzte den Stift ab. Hier in diesem Medium, dachte sie, während ihre Augen über die kahle Decke ihres Verstecks huschten, war die primäre Betrachtungsweise von Werken der Marktplatz selbst. Machte dies die Marktplatzbetreiber und deren Interfaces nicht zu wichtigen Elementen, die das Medium quasi mitkonstituierten? Und waren NFTs nicht untrennbar verbunden mit ihrem ökonomischen Trägermedium und ihrem kapitalistisch-durchdrungenen Kontext?*

Dies führte zu der formal an das Printmedium (American Legal) und Konzeptkunst (Lawrence Wiener und On Kawara) angelehnte Erscheinung zu schaffen.

Ausblick & Praxis

Die Edition habe ich Conceptual I getauft und das impliziert natürlich eine zweite oder mehrere Editionen. Aktuell habe ich nochmals ca. 60 Entwürfe, die in eine weitere Edition fliessen könnten. Technisch gesehen ist die aktuelle Edition «voll ausgeschöpft», da ihr Code die technische Maximalgrösse eines Smart Contracts erreicht hat (24KB). Bei der Auswahl der Entwürfe spielten neben den konzeptuellen Überlegungen oft auch pragmatische eine Rolle, gewisse Entwürfe stellen auf technischer Ebene zunehmend schwierige Herausforderungen, beispielsweise die Notwendigkeit von Assembly und dem direkten interagieren mit OPCODES. Dies könnte in einer zweiten Serie Platz finden. Darüber hinaus bin ich gespannt, was beim «Mainnet-Release» geschehen wird, und wie das Werk angekommen wird.

Auch in der Zukunft möchte ich die duale Praxis von reflexivem Schreiben und Schaffen weiterpflegen. Darüber hinaus verbindet diese vorliegende Arbeit viele meiner Fähigkeiten und Interessen. Ich gehe stark davon aus, dass dieser Arbeit einige ähnliche folgen werden.

Index: Conceptual I

ID	Title	Text
1	PERMISSIONLESS	Anyone can transfer this piece
2	NO EXTERNALITIES	This piece can only be held by a contract
3	OPENING HOURS	This piece has opening hours during which it exists and can be transferred
4	DEAD BEEF	This piece can only be held by an address starting with DEAD and ending with BEEF
5-6	MUTUALLY EXCLUSIVE	The two pieces of this edition are mutually exclusive. Each piece can only be owned if the other owning address has no matching digits
7	FAIR PRICE	This piece can be directly bought from the current owner by paying more than the last paid price
8	BEEF BABE	This piece can only be held by an address starting with BEEF and ending with BABE
9	SATED	This piece can only be transferred as the only transaction in a block
10	DARK	This piece can only be held by addresses containing at least 13 instances of 0 and no instances of F
11	RETRACTABLE	The artist can retract this piece at any time
12	BAD BEEF	This piece can only be held by an address starting with BAAD and ending with BEEF
13	SECRET POEM	This piece is a secret poem only know to the artist. Poem: 0xbbef8abf4b95d23c927251318c9136aca5070fdedb7ae52def0d65852181be78
14-16	SAME BLOCK	The three pieces from this edition can only be transferred in the same block
17	LIMITED USE	This piece can be transferred 7 times after which it will self destruct
18	DEAF BABE	This piece can only be held by an address starting with DEAF and ending with BABE
19	11111111111111111111	Anyone can decrease the title of this piece
20	SECRET JOKE	This piece is a secret joke only know to the artist Joke: 0x99d3578521cda2076fd39127ab261c5b61c2f266fe0477d7a5bcd55159a19570
21	LOCUS	This piece will output three spatial coordinates for the current address. It can be used to spatially relate addresses
22	PERMANENCE I	This piece exists half of the time
23	BEEF BEEF	This piece can only be held by an address starting with BEEF and ending with BEEF

24-25	CO-DEPENDENT	The two pieces of this edition are co-dependent. A piece can only be acquired if the first three and last three digits of both the owning addresses are smaller than or equal to 0xFFF = 4095
26	LIGHT	This piece can only be held by addresses containing at least 13 instances of F and no instances of 0
27	1	Anyone can increase the title of this piece
28	DEAD BABE	This piece can only be held by an address starting with DEAD and ending with BABE
29	CONTINUOUS	This piece can only be transferred to an address that has the first three digits of the previous address as its last three
30	EOA	This piece cannot be held by a contract
31	PERMANENCE II	This piece exists half of the time
32	DEAF BEEF	This piece can only be held by an address starting with DEAF and ending with BEEF
33	MAXIMALISM	This piece has no restrictions placed upon it. The sole request the artist makes is that it should not be sold to maximalists of any kind
34	COINBASE	This piece can only transferred to the block.coinbase address. As such it can only be received by validators
35	BABE BEEF	This piece can only be held by an address starting with BABE and ending with BEEF
36	BEEF FACE	This piece can only be held by an address starting with BEEF and ending with FACE
37	FEED BEEF	This piece can only be held by an address starting with FEED and ending with BEEF
38	SECRET TRUTH	This piece is a secret truth only known to the artist. Truth: 0xbb3f0eb00a365c35fe15d5df77440911e36ea2247b5ce93b4b62de3318342e70
39	CHROMATIC	This piece can only be held by addresses containing at least 1 of each hexadecimal digit
40	UNWIELDY	This piece requires burning 1 ETH to be transferred
41	BAD BABE	This piece can only be held by an address starting with BAD and ending with BABE
42	TRANSITORY OWNERSHIP	This piece will always belong to the current validator

ConceptualOne.sol

```
// SPDX-License-Identifier: AGPL-3.0-only
pragma solidity ^0.8.15;

import "./ConceptualERC721.sol";

import {ConceptStruct} from "../libraries/ConceptStruct.sol";
import {Render} from "../libraries/Render.sol";
import {Util} from "../libraries/Util.sol";

uint256 constant PERMISSIONLESS = 1;
uint256 constant NO_EXTERNALITIES = 2;
uint256 constant OPENING_HOURS = 3;
uint256 constant DEAD_BEEF = 4;
uint256 constant MUTUAL_A = 5;
uint256 constant MUTUAL_B = 6;
uint256 constant FAIR_PRICE = 7;
uint256 constant BEEF_BABE = 8;
uint256 constant SATED = 9;
uint256 constant DARK = 10;
uint256 constant RETRACTED = 11;
uint256 constant BAD_BEEF = 12;
uint256 constant EOA = 13;
uint256 constant SAMEBLOCK_I = 14;
uint256 constant SAMEBLOCK_II = 15;
uint256 constant SAMEBLOCK_III = 16;
uint256 constant LIMITED_USE = 17;
uint256 constant DEAF_BABE = 18;
uint256 constant DECREASE = 19;
uint256 constant SECRET_POEM = 20;
uint256 constant LOCUS = 21;
uint256 constant PERMANENCE_I = 22;
uint256 constant BEEF_BEEF = 23;
uint256 constant DEPENDENT_A = 24;
uint256 constant DEPENDENT_B = 25;
uint256 constant LIGHT = 26;
uint256 constant INCREASE = 27;
uint256 constant DEAD_BABE = 28;
uint256 constant CONTINUOUS = 29;
uint256 constant SECRET_JOKE = 30;
uint256 constant PERMANENCE_II = 31;
uint256 constant DEAF_BEEF = 32;
uint256 constant MAXIMALISM = 33;
uint256 constant COINBASE = 34;
uint256 constant BABE_BEEF = 35;
uint256 constant BEEF_FACE = 36;
uint256 constant FEED_BEEF = 37;
uint256 constant SECRET_TRUTH = 38;
uint256 constant CHROMATIC = 39;
uint256 constant UNWIELDY = 40;
uint256 constant BAD_BABE = 41;
uint256 constant TRANSITORY_OWNERSHIP = 42;
```

```
contract ConceptualOne is ConceptualERC721 {

   /*////////////////////////////////////////////////////////////////
                            Constructor
   ////////////////////////////////////////////////////////////////*/

   constructor() ConceptualERC721("Conceptual I", "CONCEPT") {
       _mint(block.coinbase, TRANSITORY_OWNERSHIP);
       totalSupply++;
   }

   /*////////////////////////////////////////////////////////////////
                            Tokens / Mint
   ////////////////////////////////////////////////////////////////*/

   uint256 public totalSupply = 0;
   uint256 public maxMint = 42;
   uint256 constant public price = 0.1 ether;

   function mint(uint256 _tokenId) public payable {
       require(_tokenId > 0 && _tokenId <= maxMint);
       uint256 mintPrice = (_tokenId == UNWIELDY ? (1 ether + price) : price);
       require(msg.value == mintPrice);
       _mint(msg.sender, _tokenId);
       totalSupply++;
   }

   function setMaxMint(uint256 _maxMint) external onlyArtist {
       maxMint = _maxMint;
   }

   function withdraw(address payable _to) public onlyArtist {
       (bool success,) = _to.call{value: address(this).balance}("");
       require(success);
   }

   /*////////////////////////////////////////////////////////////////
                            Concept Data
   ////////////////////////////////////////////////////////////////*/

   mapping (uint256 => ConceptStruct.Concept) concepts;
   mapping (uint256 => uint256) tokenIdToConcept;

   function setConceptData(ConceptStruct.Concept[] memory _concepts) public
onlyArtist {
       for (uint i = 0; i < _concepts.length; i++) {
           ConceptStruct.Concept memory concept = _concepts[i];
           uint256 conceptId = concept._editionTokenRangeStart;
           concepts[conceptId] = concept;
           for (uint j = concept._editionTokenRangeStart; j <
concept._editionTokenRangeStart + concept._editionSize; j++) {
               tokenIdToConcept[j] = conceptId;
```

```
            }
        }
    }

    /*////////////////////////////////////////////////////////////////
                            Custom State
    ////////////////////////////////////////////////////////////////*/

    mapping (uint256 => uint256) sameBlockAt;
    mapping (uint256 => address) sameBlockTo;
    uint256 public fairPrice;
    uint96 public increasable = 1;
    uint96 public decreasable = 111111111111111111111111111;
    uint256 public transfersLeft = 7;

    /*////////////////////////////////////////////////////////////////
                            Custom Methods
    ////////////////////////////////////////////////////////////////*/

    function buy(uint256 _tokenId) public payable {
        if (_tokenId == FAIR_PRICE) {
            if (_ownerOf[_tokenId] == address(0)) {
                fairPrice = msg.value;
                _mint(msg.sender, FAIR_PRICE);
                return;
            } else if (msg.value > fairPrice) {
                fairPrice = msg.value;
                _transferFromInternalNoHooksAndChecks(_ownerOf[_tokenId],
msg.sender, FAIR_PRICE);
                return;
            }
        }
        revert();
    }

    function increase() public payable {
        increasable++;
    }

    function decrease() public payable {
        decreasable++;
    }

    function retract() public onlyArtist {
        _transferFromInternalNoHooksAndChecks(_ownerOf[RETRACTED], artist(),
RETRACTED);
    }

    function isOpenHours() public view returns (bool) {
        uint256 daytime = block.timestamp % (24 * 3600);
        return (daytime >= 9 * 3600) && (daytime <= 17 * 3600); // 9-5 UTC
    }
```

```
   // This piece will output three spatial coordinates for the current address.
   // It can be used to spatially relate addresses.
   function whereIs(address _address) public pure returns (uint256 x, uint256 y ,
uint256 z) {
        uint256 addressNumber = uint256(keccak256(abi.encodePacked(_address)));
        x = addressNumber % 10e4;
        y = (addressNumber >> 8) % 10e4;
        z = (addressNumber >> 16) % 10e3;
   }

   /*////////////////////////////////////////////////////////////
                          Hooks and Overrides
   ////////////////////////////////////////////////////////////*/

   function ownerOf(uint256 id) public view override returns (address) {
        if (id == TRANSITORY_OWNERSHIP) {
            return block.coinbase;
        } else if (
            (id == PERMANENCE_I  && block.number % 2 == 0) ||
            (id == PERMANENCE_II && block.number % 2 == 1)
        ) {
            revert("Half of the time");
        } else if (id == OPENING_HOURS) {
            require(isOpenHours(), "Currently closed");
        }
        return super.ownerOf(id);
   }

   function balanceOf(address owner) public view override returns (uint256) {
        if (owner == block.coinbase) {
            return 1 + _balanceOf[owner];
        }
        return super.balanceOf(owner);
   }

   function transferFrom(
        address from,
        address to,
        uint256 id
   ) public virtual override payable {
        if (id == PERMISSIONLESS) {
            _transferFromInternal(from, to, id);
        } else {
            super.transferFrom(from, to, id);
        }
   }

   function _afterTokenTransfer(
        address from,
        address to,
        uint256 tokenId
   ) internal override {}
```

```
   function _beforeTokenTransfer(
       address from,
       address to,
       uint256 tokenId
   ) internal override returns (bool) {
       if (tokenId == SATED) {
           // This piece can only be transferred as the only transaction in a block
           uint256 waste; for (uint256 i = 0; i < 159160; i++) waste++;
       } else if (
           (tokenId == PERMANENCE_I  && block.number % 2 == 0) ||
           (tokenId == PERMANENCE_II && block.number % 2 == 1)
       ) {
           revert("Half of the time");
       } else if (tokenId == TRANSITORY_OWNERSHIP) {
           require(to == block.coinbase || _ownerOf[TRANSITORY_OWNERSHIP] !=
address(0));
       } else if (tokenId == OPENING_HOURS) {
           require(isOpenHours(), "Currently closed");
       } else if (tokenId == DEAD_BEEF || tokenId == BEEF_BEEF || tokenId ==
FEED_BEEF || tokenId == DEAF_BEEF
               || tokenId == BAD_BEEF  || tokenId == BEEF_FACE || tokenId ==
BABE_BEEF || tokenId == BEEF_BABE
               || tokenId == BAD_BABE  || tokenId == DEAF_BABE || tokenId ==
DEAD_BABE
       ) {
           // This piece can only be held by an address starting with DEAD and
ending with BEEF
           bytes20 addr = bytes20(to);
           if (
               (
                   (tokenId == DEAD_BEEF && (addr &
hex"ffff0000000000000000000000000000000000ffff" ==
hex"dead0000000000000000000000000000000000beef"))
                   ||
                   (tokenId == BEEF_BEEF && (addr &
hex"ffff0000000000000000000000000000000000ffff" ==
hex"beef0000000000000000000000000000000000beef"))
                   ||
                   (tokenId == FEED_BEEF && (addr &
hex"ffff0000000000000000000000000000000000ffff" ==
hex"feed0000000000000000000000000000000000beef"))
                   ||
                   (tokenId == DEAF_BEEF && (addr &
hex"ffff0000000000000000000000000000000000ffff" ==
hex"deaf0000000000000000000000000000000000beef"))
                   ||
                   (tokenId == BAD_BEEF && (addr &
hex"fff00000000000000000000000000000000000ffff" ==
hex"bad0000000000000000000000000000000000beef"))
                   ||
                   (tokenId == BEEF_FACE && (addr &
hex"ffff0000000000000000000000000000000000ffff" ==
hex"beef0000000000000000000000000000000000face"))
```

```
                    ||
                    (tokenId == BABE_BEEF && (addr &
hex"ffff0000000000000000000000000000ffff" ==
hex"babe0000000000000000000000000000beef"))
                    ||
                    (tokenId == BEEF_BABE && (addr &
hex"ffff0000000000000000000000000000ffff" ==
hex"beef0000000000000000000000000000babe"))
                    ||
                    (tokenId == BAD_BABE && (addr &
hex"fff00000000000000000000000000000ffff" ==
hex"bad00000000000000000000000000000babe"))
                    ||
                    (tokenId == DEAF_BABE && (addr &
hex"ffff0000000000000000000000000000ffff" ==
hex"deaf0000000000000000000000000000babe"))
                    ||
                    (tokenId == DEAD_BABE && (addr &
hex"ffff0000000000000000000000000000ffff" ==
hex"dead0000000000000000000000000000babe"))
                ) == false
            ) {
                revert('No beef');
            }

        } else if (tokenId >= SAMEBLOCK_I && tokenId <= SAMEBLOCK_III) {
            // The three pieces from this edition can only be transferred in the
same block
            uint256 dependentIdA = tokenId == SAMEBLOCK_I ? SAMEBLOCK_II :
SAMEBLOCK_I;
            uint256 dependentIdB = tokenId == SAMEBLOCK_I ? SAMEBLOCK_III : tokenId
== SAMEBLOCK_II ? SAMEBLOCK_III : SAMEBLOCK_II;
            if (sameBlockAt[dependentIdA] == block.number &&
sameBlockAt[dependentIdB] == block.number) {
                _ownerOf[dependentIdA] == address(0)
                    ? _mintNoHooks(sameBlockTo[dependentIdA], dependentIdA)
                    : _transferFromInternalNoHooksAndChecks(_ownerOf[dependentIdA],
sameBlockTo[dependentIdA], dependentIdA);
                _ownerOf[dependentIdB] == address(0)
                    ? _mintNoHooks(sameBlockTo[dependentIdB], dependentIdB)
                    : _transferFromInternalNoHooksAndChecks(_ownerOf[dependentIdB],
sameBlockTo[dependentIdB], dependentIdB);
                return true;
            } else {
                sameBlockAt[tokenId] = block.number;
                sameBlockTo[tokenId] = to;
                return false;
            }
        } else if (tokenId == DEPENDENT_A || tokenId == DEPENDENT_B) {
            // The two pieces of this edition are co-dependent. A piece can only be
acquired if the sum of the
            // first three and of the last three digits of both the owning addresses
are smaller than or equal
```

```
            // to 0xFFF = 4095 respectively
            uint256 dependentId = tokenId == DEPENDENT_A ? DEPENDENT_B :
DEPENDENT_A;
            if (
                (
                    uint160(_ownerOf[dependentId]) % 4096 + uint160(to) % 4096 <
4096
                    &&
                    uint160(bytes20(_ownerOf[dependentId]) >> 148) % 4096 +
uint160(bytes20(to) >> 148) % 4096 < 4096
                ) == false
            ) {
                revert('Codependent');
            }
        } else if (tokenId == MUTUAL_A || tokenId == MUTUAL_B) {
            // The two pieces of this edition are mutually exclusive. Each piece can
only be
            // owned if the other owning address has no matching digits
            uint256 dependentId = tokenId == MUTUAL_A ? MUTUAL_B : MUTUAL_A;
            bytes20 bytesDependent = bytes20(_ownerOf[dependentId]);
            bytes20 bytesTo = bytes20(to);
            for (uint i = 0; i < 40; i++) {
                if (uint160(bytesDependent >> i * 4) % 16 == uint160(bytesTo >> i *
4) % 16) {
                    revert('Mutually exclusive');
                }
            }
        } else if (tokenId == LIGHT) {
            // This piece can only be held by addresses containing at least 13
instances of F and no instances of 0
            bytes20 bytesTo = bytes20(to);
            uint16 count;
            for (uint i ; i < 40; i++) {
                if (uint160(bytesTo >> i * 4) % 16 == 0) break;
                if (uint160(bytesTo >> i * 4) % 16 == 15) count++;
                if (count > 13) return true;
            }
            revert();
        } else if (tokenId == DARK) {
            // This piece can only be held by addresses containing at least 13
instances of 0 and no instances of F
            bytes20 bytesTo = bytes20(to);
            uint16 count;
            for (uint i ; i < 40; i++) {
                if (uint160(bytesTo >> i * 4) % 16 == 15) break;
                if (uint160(bytesTo >> i * 4) % 16 == 0) count++;
                if (count > 13) return true;
            }
            revert();
        } else if (tokenId == CHROMATIC) {
            // This piece can only be held by addresses containing at least 1 of
each hexadecimal digit
            bytes20 bytesTo = bytes20(to);
```

```
            uint256 bitmap;
            for (uint256 i; i < 40; i++) {
                bitmap |= (1 << (uint160(bytesTo >> i * 4) % 16));
                if (bitmap == 65535) return true;
            }
            revert();
        } else if (tokenId == NO_EXTERNALITIES) {
            // This piece can only be held by a contract
            uint size;
            assembly { size := extcodesize(to) }
            require(size > 0);
        } else if (tokenId == EOA) {
            // This piece cannot be held by a contract
            uint size;
            assembly { size := extcodesize(to) }
            require(size == 0);
            require(tx.origin == msg.sender);
        } else if (tokenId == LIMITED_USE) {
            // This piece can only be transferred 7 times after which it will self
destruct
            if (transfersLeft > 0) {
                transfersLeft--;
            } else {
                // bye
                _transferFromInternalNoHooksAndChecks(_ownerOf[LIMITED_USE],
address(0xdEaD), LIMITED_USE);
                return false; // don't transfer
            }
        } else if (tokenId == UNWIELDY) {
            // This piece requires burning 1 ETH to be transferred.
            require(msg.value >= 1 ether && msg.value <= 1 ether + price);
            (bool success,) = address(0).call{value: 1 ether}(""); // good bye
            require (success);
        } else if (tokenId == COINBASE) {
            // This piece can only transferred to the block.coinbase address.
            // As such it can only be received by validators
            require(to == block.coinbase);
        } else if (tokenId == CONTINUOUS) {
            // This piece can only be transferred to an address that has the
            // first three digits of the previous address as its last three
            bytes20 bytesFrom = bytes20(_ownerOf[CONTINUOUS]);
            bytes20 bytesTo = bytes20(to);
            for (uint i = 0; i < 3; i++) {
                if (uint160(bytesFrom >> (i) * 4) % 16 != uint160(bytesTo >> (40 -
(3 - i)) * 4) % 16) {
                    revert('Not Continuous');
                }
            }
        }

        return true;
    }
```

```
    ////////////////////////////////////////////////////////////////////////
    // Metadata
    ////////////////////////////////////////////////////////////////////////

    function renderSVG(uint256 _tokenId) external view returns (string memory) {
        ConceptStruct.Concept memory concept = _getConceptFromTokenId(_tokenId);
        return Render.renderSVG(_tokenId, concept, font);
    }

    function renderSVGBase64(uint256 _tokenId) external view returns (string memory)
{
        ConceptStruct.Concept memory concept = _getConceptFromTokenId(_tokenId);
        return Render.renderSVGBase64(_tokenId, concept, font);
    }

    function tokenURI(uint256 _tokenId) public view override returns (string memory)
{
        if (
            (_tokenId == PERMANENCE_I  && block.number % 2 == 0) ||
            (_tokenId == PERMANENCE_II && block.number % 2 == 1)
        ) {
            revert("Half of the time");
        } else if (_tokenId == OPENING_HOURS) {
            require(isOpenHours(), "Currently closed");
        }
        require(_ownerOf[_tokenId] != address(0), "NOT_MINTED");
        ConceptStruct.Concept memory concept = _getConceptFromTokenId(_tokenId);
        return Render.tokenURI(_tokenId, concept, font);
    }

    function _getConceptFromTokenId(uint256 _tokenId) private view returns
(ConceptStruct.Concept memory) {
        uint256 conceptId = tokenIdToConcept[_tokenId];
        ConceptStruct.Concept memory concept = concepts[conceptId];

        if (_tokenId == LIMITED_USE) {
            bytes32[] memory statusText = new bytes32[](1);
            statusText[0] = bytes32(abi.encodePacked("Transfers Left: ",
Util.uint256ToString(uint256(transfersLeft))));
            concept._statusText = statusText;
        } else if (_tokenId == INCREASE) {
            concept._title =
bytes32(abi.encodePacked(Util.uint256ToString(uint256(increasable))));
        } else if (_tokenId == DECREASE) {
            concept._title =
bytes32(abi.encodePacked(Util.uint256ToString(uint256(decreasable))));
        } else if (_tokenId == OPENING_HOURS) {
            bytes32[] memory statusText = new bytes32[](1);
            statusText[0] = bytes32(abi.encodePacked("Currently ", isOpenHours() ?
"Open" : "Closed"));
            concept._statusText = statusText;
        } else if (_tokenId == LOCUS) {
            bytes32[] memory statusText = new bytes32[](3);
```

```
            uint256 x; uint256 y; uint256 z;
            (x,y,z) = whereIs(_ownerOf[_tokenId]);
            statusText[0] = bytes32(abi.encodePacked(
                Util.uint256ToString(x), ", ",
                Util.uint256ToString(y), ", ",
                Util.uint256ToString(z)
            ));
            concept._statusText = statusText;
        }

        return concept;
    }

    ////////////////////////////////////////////////////////////////////////
    // Font
    ////////////////////////////////////////////////////////////////////////

    string private font = "data:application/font-woff2;charset=utf-8;base64,...";

}
```

ConceptualERC721.sol

```
// SPDX-License-Identifier: AGPL-3.0-only
pragma solidity >=0.8.0;

import "@openzeppelin/contracts/access/Ownable.sol";

/// @notice Adopted from Solmate
(https://github.com/transmissions11/solmate/blob/main/src/tokens/ERC721.sol)
abstract contract ConceptualERC721 is Ownable {

   /*//////////////////////////////////////////////////////////////
                         ERC721 STORAGE
   //////////////////////////////////////////////////////////////*/

   string public name;

   string public symbol;

   mapping(uint256 => address) public getApproved;

   mapping(address => mapping(address => bool)) public isApprovedForAll;

   mapping(uint256 => address) internal _ownerOf;

   mapping(address => uint256) internal _balanceOf;

   /*//////////////////////////////////////////////////////////////
                                EVENTS
   //////////////////////////////////////////////////////////////*/

   event Transfer(address indexed from, address indexed to, uint256 indexed id);

   event Approval(address indexed owner, address indexed spender, uint256 indexed
id);

   event ApprovalForAll(address indexed owner, address indexed operator, bool
approved);

   /*//////////////////////////////////////////////////////////////
                                VIEW
   //////////////////////////////////////////////////////////////*/

   function tokenURI(uint256 id) public view virtual returns (string memory);


   function ownerOf(uint256 id) public view virtual returns (address) {
       address owner = _ownerOf[id];
       require(owner != address(0), "NOT_MINTED");
       return owner;
   }

   function balanceOf(address owner) public view virtual returns (uint256) {
```

```
        require(owner != address(0), "ZERO_ADDRESS");
        return _balanceOf[owner];
    }

    /*//////////////////////////////////////////////////////////////
                                VANITY
    //////////////////////////////////////////////////////////////*/

    function artist() public view returns (address) {
        return owner();
    }

    modifier onlyArtist {
        require(msg.sender == artist());
        _;
    }

    /*//////////////////////////////////////////////////////////////
                              CONSTRUCTOR
    //////////////////////////////////////////////////////////////*/

    constructor(string memory _name, string memory _symbol) {
        name = _name;
        symbol = _symbol;
    }

    /*//////////////////////////////////////////////////////////////
                             ERC721 LOGIC
    //////////////////////////////////////////////////////////////*/

    function approve(address spender, uint256 id) public virtual {
        address owner = _ownerOf[id];
        require(msg.sender == owner || isApprovedForAll[owner][msg.sender],
"NOT_AUTHORIZED");

        getApproved[id] = spender;

        emit Approval(owner, spender, id);
    }

    function setApprovalForAll(address operator, bool approved) public virtual {
        isApprovedForAll[msg.sender][operator] = approved;

        emit ApprovalForAll(msg.sender, operator, approved);
    }

    function transferFrom(
        address from,
        address to,
        uint256 id
    ) public virtual payable {
        require(
```

```
            msg.sender == from || isApprovedForAll[from][msg.sender] || msg.sender
== getApproved[id],
            "NOT_AUTHORIZED"
        );

        _transferFromInternal(from, to, id);
    }

    function _transferFromInternal(
        address from,
        address to,
        uint256 id
    ) internal virtual {
        require(to != address(0), "INVALID_RECIPIENT");
        require(from == _ownerOf[id], "WRONG_FROM");

        bool doTransfer = _beforeTokenTransfer(from, to, id);
        if (!doTransfer) return;

        // Underflow of the sender's balance is impossible because we check for
        // ownership above and the recipient's balance can't realistically overflow.
        unchecked {
            _balanceOf[from]--;

            _balanceOf[to]++;
        }

        _ownerOf[id] = to;

        delete getApproved[id];

        emit Transfer(from, to, id);

        _afterTokenTransfer(from, to, id);
    }

    function _transferFromInternalNoHooksAndChecks(
        address from,
        address to,
        uint256 id
    ) internal virtual {
        require(to != address(0), "INVALID_RECIPIENT");

        // Underflow of the sender's balance is impossible because we check for
        // ownership above and the recipient's balance can't realistically overflow.
        unchecked {
            _balanceOf[from]--;

            _balanceOf[to]++;
        }

        _ownerOf[id] = to;
```

```
        delete getApproved[id];

        emit Transfer(from, to, id);
    }

    function safeTransferFrom(
        address from,
        address to,
        uint256 id
    ) public virtual payable {
        transferFrom(from, to, id);

        require(
            to.code.length == 0 ||
                ERC721TokenReceiver(to).onERC721Received(msg.sender, from, id, "")
==
                ERC721TokenReceiver.onERC721Received.selector,
            "UNSAFE_RECIPIENT"
        );
    }

    function safeTransferFrom(
        address from,
        address to,
        uint256 id,
        bytes calldata data
    ) public virtual payable {
        transferFrom(from, to, id);

        require(
            to.code.length == 0 ||
                ERC721TokenReceiver(to).onERC721Received(msg.sender, from, id, data)
==
                ERC721TokenReceiver.onERC721Received.selector,
            "UNSAFE_RECIPIENT"
        );
    }

    /*//////////////////////////////////////////////////////////////
                              ERC165 LOGIC
    //////////////////////////////////////////////////////////////*/

    function supportsInterface(bytes4 interfaceId) public view virtual returns
(bool) {
        return
            interfaceId == 0x01ffc9a7 || // ERC165 Interface ID for ERC165
            interfaceId == 0x80ac58cd || // ERC165 Interface ID for ERC721
            interfaceId == 0x5b5e139f; // ERC165 Interface ID for ERC721Metadata
    }

    /*//////////////////////////////////////////////////////////////
                        INTERNAL MINT/BURN LOGIC
    //////////////////////////////////////////////////////////////*/
```

```
function _mint(address to, uint256 id) internal virtual {
    require(to != address(0), "INVALID_RECIPIENT");
    require(_ownerOf[id] == address(0), "ALREADY_MINTED");

    bool doTransfer =_beforeTokenTransfer(address(0), to, id);
    if (!doTransfer) return;

    // Counter overflow is incredibly unrealistic.
    unchecked {
        _balanceOf[to]++;
    }

    _ownerOf[id] = to;

    emit Transfer(address(0), to, id);

    _afterTokenTransfer(address(0), to, id);
}

function _mintNoHooks(address to, uint256 id) internal virtual {
    require(to != address(0), "INVALID_RECIPIENT");
    require(_ownerOf[id] == address(0), "ALREADY_MINTED");

    // Counter overflow is incredibly unrealistic.
    unchecked {
        _balanceOf[to]++;
    }

    _ownerOf[id] = to;

    emit Transfer(address(0), to, id);
}

function _beforeTokenTransfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual returns (bool) {}

function _afterTokenTransfer(
    address from,
    address to,
    uint256 tokenId
) internal virtual {}

/*//////////////////////////////////////////////////////////////
                    INTERNAL SAFE MINT LOGIC
//////////////////////////////////////////////////////////////*/

function _safeMint(address to, uint256 id) internal virtual {
    _mint(to, id);
```

```
        require(
            to.code.length == 0 ||
                ERC721TokenReceiver(to).onERC721Received(msg.sender, address(0), id,
"") ==
                ERC721TokenReceiver.onERC721Received.selector,
            "UNSAFE_RECIPIENT"
        );
    }

    function _safeMint(
        address to,
        uint256 id,
        bytes memory data
    ) internal virtual {
        _mint(to, id);

        require(
            to.code.length == 0 ||
                ERC721TokenReceiver(to).onERC721Received(msg.sender, address(0), id,
data) ==
                ERC721TokenReceiver.onERC721Received.selector,
            "UNSAFE_RECIPIENT"
        );
    }
}

/// @notice A generic interface for a contract which properly accepts ERC721
tokens.
/// @author Solmate
(https://github.com/transmissions11/solmate/blob/main/src/tokens/ERC721.sol)
abstract contract ERC721TokenReceiver {
    function onERC721Received(
        address,
        address,
        uint256,
        bytes calldata
    ) external virtual returns (bytes4) {
        return ERC721TokenReceiver.onERC721Received.selector;
    }
}
```