Marc Bendt: .HYP – A digital wildgrowth

Diplomarbeit Kommunikationsdesign
Hochschule für Gestaltung Karlsruhe, 2024

****English version below****

Der Name des Programms .HYP leitet sich vom Namen der Zellen von Pilzen ab, den sogenannten Hyphen. Ein bißchen wie das Wurzelwerk von Pflanzen breiten diese sich unterirdisch aus und dienen als Werkzeuge für Wasser- und Nährstoffaufnahme von Pilzen. Der Titel des Projektes basiert außerdem auf der Endung der Unterprogramme, welche das Hauptrogramm einleitet.

.HYP ist eine Desktopanwendung, dessen Ziel es ist, in ein Dateisystem hineinzuwachsen und dort Teile von Dokumenten zu konsumieren, um daraus Energie zum Weiterwachsen zu ziehen. Dabei imitiert es die Funktionen von Pilzen, welche als Disposer, also Resteverwerter der Natur, fungieren und solche Reste zersetzen und für sich in Energie umwandeln. Der Lebensraum für das Programm .HYP sind die Ordnersysteme und die darin enthaltenen Dateien, welche das Programm umwandeln und Teile daraus konsumieren kann. Ziel des Programmes ist es, genug Energie und Datenmaterial zu sammeln, um daraus final eine Art digitalen Fruchtkörper zu bilden, welchen man als Zuschauer*in betrachten kann. Wie bei unterirdischen Hyphensystemen von Pilzen ist der Rest des Programms für den Menschen von außen nicht ersichtlich.

Für das Ausbreiten und Konsumieren des digitalen Pilzes teilt sich das Programm in Unterprogramme auf, welche ähnlich wie die Hyphen der Pilze verschiedene Aufgaben übernehmen:

discovery.HYP
Dieses Teilprogramm von .HYP dient zur Raumerschließung und Ausbreitung des digitalen Pilzes und erkennt, wie viele Ordner und Dateien sich in seiner Umgebung befinden. Auf Basis dieser Zahlen kann das Programm errechnen, ob es sich auf Konsum von Dateien oder Ausbreitung in Unterordner fokussieren soll.

collector.HYP
Die collector hyphe ist für die Umwandlung und den Konsum von Dateien zuständig. Sie erkennt Bilder und kann diese nach gesättigten Pixeln durchsuchen und diese Pixel dann speichern oder in Energie umwandeln. Alle Dokumente, die kein Bildformat sind, werden in einem vorherigen Schritt in Bilder? umgewandelt und dann konsumiert. Nach dem Sammeln der Pixel aus den Bildern werden diese Pixel dann in einem quadratischen Bild zwischengelagert. Dieses wird als fruitPart_X in den jeweiligen Ordnern der collector-Hyphen abgespeichert und zu einem späteren Zeitpunkt als Baumaterial für den Fruchtkörper des Programms verwendet.

safe.HYP
Dieser Teil des Programms speichert die Energie, welche die collector.HYP aus einem Dokument gezogen hat, und verteilt diese dann für den gesamten digitalen Pilz.

mainRoot.HYP
Organisiert und geleitet werden die Aufgaben des digitalen Pilzes von einem übergeordneten Programm, der mainRoot.HYP. Dieses Programm sammelt die Energie von safe.HYP und weiß, wo sich die gebildeten fruitParts und Hyphen in den Ordnern befinden.

Hat der Pilz genug Energie aus anderen Dokumenten gespeichert und genug Pixel in den fruitParts gesammelt, fängt .HYP an, einen Fruchtkörper zu bilden, welcher den Fruchtkörper von Pilzen aus der Natur nachahmen soll. Dieser digitale Fruchtkörper bedient sich an den gesammelten Pixeln in den fruitParts-Dateien und wandelt diese dann in verschiedene Formen und Muster um, welche man dann im Interface des Programmes und im Projektionsraum betrachten kann.

English version:

The name of the program .HYP is derived from hyphae, the branching filaments that make up the mycelium of a fungus. Much like the root systems of plants, these spread underground and serve as tools for the fungus to absorb water and nutrients. The project title is also based on the file extensions of the subprograms listed below, which the main program initiates; these extensions are all .HYP.

.HYP is a desktop application designed to "grow" into a file system, consuming parts of documents to gain the energy required for further expansion. In doing so, it mimics the function of fungi, which act as nature's decomposers—breaking down waste and converting it into energy. The "habitat" for the .HYP program consists of folder systems and the files within them, which the program transforms and consumes.

The ultimate goal of the program is to gather enough energy and data to eventually form a digital fruiting body (sporocarp) for the viewer to observe. As with the underground hyphal systems of real fungi, the rest of the program remains hidden from the human user. To spread and consume, the program divides itself into subprograms that perform different tasks, similar to biological hyphae:

discovery.HYP
This subprogram is used for spatial exploration and the expansion of the digital fungus. It detects how many folders and files are in its vicinity. Based on these numbers, the program calculates whether to focus on consuming files or spreading into subdirectories.

collector.HYP
The collector hypha is responsible for the transformation and consumption of files. It identifies images and scans them for saturated pixels, which it then saves or converts into energy. Any documents that are not in an image format are converted into images in a preliminary step before being consumed. Once the pixels are gathered, they are temporarily stored in a square image file. These are saved as fruitPart_X in the respective folder of the collector hyphae and serve as "building blocks" for the program's fruiting body at a later stage.

safe.HYP
This file stores the energy that the collector.HYP has extracted from documents and distributes it throughout the entire digital fungus.

mainRoot.HYP
The tasks of the digital mushroom are organized and managed by a parent program, mainRoot.HYP. This program collects energy from safe.HYP and knows where the generated fruitParts and hyphae are located in the folders.

Once the mushroom has stored enough energy from other documents and collected enough pixels in the fruitParts, .HYP begins to form a fruiting body designed to mimic the fruiting bodies of mushrooms found in nature. This digital fruiting body draws upon the collected pixels in the fruitParts files and then transforms them into various shapes and patterns, which can then be viewed in the program's interface and in the projection space.