

AvantZero

Random data-generating algorithm for (experimental)
Quake III Machinima (post)-production

TECHNICAL PERSPECTIVES

Veenstra, J.P.
Machinima Artist, Software Developer and Lecturer

August 2025

Table of Contents

Chapter 1: Background	p. 7
Chapter 2: Technology Stack	p. 14
Chapter 3: Goals and Objectives	p. 19
Chapter 4: Data Structures & System Workflow	p. 23
Chapter 5: Error Handling	p. 51
Chapter 6: Unit Testing	p. 55
Chapter 7: Development Sprints	p. 57
Chapter 8: Deployment & Future Development	p. 62
Chapter 9: Limitations	p. 63
Chapter 10: Looking Beyond: The Avant Algorithm	p. 64
References	p. 66

List of Terms and Abbreviations

Python

A versatile, high-level, general-purpose programming language; well known for its readability and expansive toolset for machine learning, scripting, algorithms, artificial intelligence, web development, stand-alone web applications and data analysis.

CLI

Abbreviation standing for '*Command Line Interface*'; a piece of software that requires text commands for command/software initialization and execution instead of a graphical interface (containing menu's, buttons and other graphical representations)

Algorithm

Software containing a specific and custom-made set of rules to get data such as media, text and the like from point A (input) towards a desired point B (output).

Machinima

Combination of the words 'machine' and 'cinema'; a popular artform for participatory media creation since the late 1990's/early 2000's. Literal meaning: the creation of digital films inside videogame engines (often by utilizing videogame assets, code adjustments, code execution and game modification programs)¹

¹ Oxford Dictionary (n.d.). *Machinima*. In *Oxford English Dictionary*. Retrieved July 16, 2025, from https://www.oed.com/dictionary/machinima_n?tl=true

Avant Garde

Popular artform that develops new experimental concepts or techniques; often associated with the arts²

Dominion Algorithm

A random data generator for experimental Quake III machinima with limited functionality. Predecessor of the Avant algorithm. Initially created in late 2022 – early 2023 with the JavaScript language. Now considered legacy and defunct.³

Proof of Concept

A version of specific software with limited functionality and scope to prove the feasibility of a concept or idea. AvantZero is considered a ‘proof of concept’ and only provides limited functionality compared to Avant – which is the full version of AvantZero - and is currently, as of July 2025, still in active development.

Quake III Arena

Videogame released in December 1999 by game-developer *ID Software*⁴. Quake III Arena is a multiplayer-based first-person shooter, in which the player is able to compete (often during fast-paced gameplay) against computer-controlled enemies (bots) or other online players in arena-based levels (better known as ‘maps’) throughout a multitude of different game modes, such as, but not limited to: team deathmatch, 1 on 1 duels and Capture the Flag.

² Merriam-Webster. (n.d.). *Avant-garde*. In *Merriam-Webster.com dictionary*. Retrieved July 17, 2025, from <https://www.merriam-webster.com/dictionary/avant-garde>

³ Veenstra, J.P. (2024). Dominion Algorithm [Bitbucket Repository]. *Bitbucket*. Retrieved July 17, 2025, from: <https://bitbucket.org/appov/dominion/src/main/>

⁴ Altar of Gaming (n.d.). *Quake III Arena*. Altar of Gaming. Retrieved July 16, 2025, from <https://altarofgaming.com/game/quake-iii-arena/>

NLE

Abbreviation of: Non-Linear Editor; Software suite used for video-editing.

GUI

Abbreviation of '*Graphical User Interface*'; a piece of software that contains a graphics-based interface with buttons, (colorful) text, navigable menus and other forms of functionality for easy reference and execution of tasks.

Q3MME

Abbreviation of '*Quake III Movie Maker's Edition*'. A modification made for the game Quake III Arena by the users HMage, ent, auri and CaNaBiS. Q3MME allows the users to capture footage within the ID Tech 3 engine from recorded gameplay and turn them into video files or image sequences for use in machinima or game art projects. The mod contains several advanced functions such as Multi-View support (allowing to see multiple point of views at once), Depth of Field generation, custom export parameters and the implementation of motion blur.⁵

CRUD

An acronym (also known as C.R.U.D.) standing for 'Create, Read, Update, Delete': four basic operations in computing and data management.⁶

⁵ entdark. (n.d.). *q3mme* [GitHub repository]. GitHub. Retrieved July 16, 2025, from: <https://github.com/entdark/q3mme>

⁶ Codecademy (n.d.). *What is CRUD?* Codecademy. Retrieved July 16, 2025, from: <https://www.codecademy.com/article/what-is-crud-explained>

API

Abbreviation standing for ‘Application Programming Interface’: a piece of standalone software or set of rules/protocols that allow applications to communicate or exchange functionality and/or data⁷.

OOP

Abbreviation standing for ‘Object Oriented Programming’: a programming paradigm focused on the use of classes and objects (instances of classes) instead of function-based code.

⁷ Goodwin, M. (2024). *What is an API (Application Programming Interface)?* IBM. Retrieved July 18, 2025, from: <https://www.ibm.com/think/topics/api>

Chapter 1 – Background

AvantZero is a Python-based tool designed to simplify and streamline the post-production process of randomized experimental machinima films with an emphasis placed on Quake III machinima. It was initially conceived as an experimental edit-decision randomizing algorithm with very basic functionality in late 2022 – early 2023 as the ‘Dominion’ algorithm. Written in Node.js and JavaScript over the course of a few weeks and released on NPM afterwards, Dominion was quickly considered defunct and legacy and more of a prototype for newer, upcoming algorithms due to its severe lack of automation capabilities and generation of vast amounts of data, which still had to be interpreted, sorted and used by hand. It merely outputted randomized edit decision data and production-data, to which the editor should adhere to during both the capturing and editing process. At that given point in time, Dominion failed to automate anything for the user.

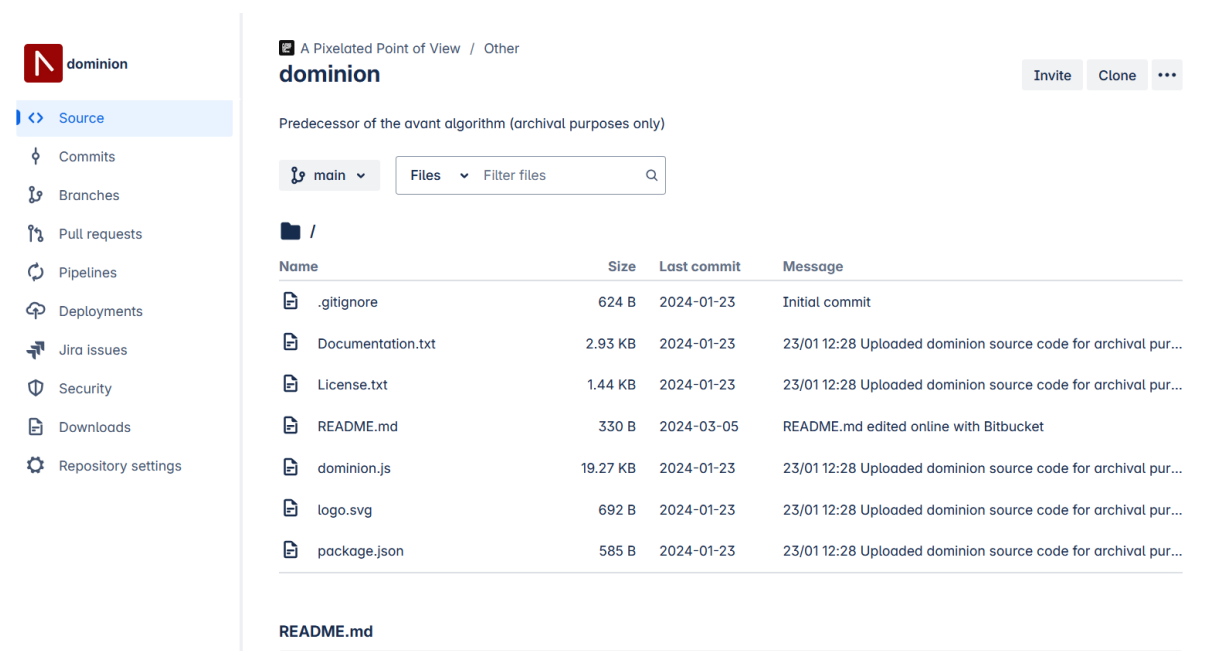


Figure 1: The bitbucket repository containing the Dominion algorithm⁸

⁸ Veenstra, J.P. (2025). *Dominion Algorithm Repository* [Image].

The Dominion algorithm was originally developed as a both a production and post-production tool for the machinima label ‘A Pixelated Point of View’. Its main functionality revolved around randomly deciding how several experimental and randomized compositions should be captured inside of ID Software’s ID Tech 3 engine (with the Quake III MovieMaker’s Edition mod – hereafter referred to as Q3MME) and how all pieces of footage should be assembled on the NLE timeline. In a sense, it presented a missing link inside an artistic, poetic and collaborative process between humans and machines. The main line of thought behind the Dominion algorithm was that it would make final decisions regarding the visual aspect of the film, while the user remained full control and sole responsibility for the auditory aspect of the film.

At its earliest stage of usability, Dominion painfully pointed out that its level of quality was too low to be used for any useful production work. Much of the work still had to be conducted by hand, the data was not properly sorted and interpreted by the algorithm; only adding more work for the user. It merely helped the user with creating a tangible copy of production and post-production data, yet the data was often spread over numerous pages and proved to be often too extensive for any form of quick reference. Each of the steps that had to be followed afterwards still had to be conducted manually; thus, sparking the idea for the creation of a better, smarter and much more polished version of the Dominion algorithm by placing emphasis on functionality, broader usability and automation, while still embracing the core idea of ‘*randomized video (post)-production*’.

It was at that moment in early 2024 that *Avant* was born: the successor to the Dominion algorithm. Avant’s main functionality revolves around the generation of random, experimental Quake III Arena compositions while automating much of the production and post-production process. It can create all the required project files for the Quake III Movie Maker’s Edition mod, a simple executable bat file to start the entire capturing process and finally generate XML/EDL files required for the NLE to properly read and export the randomized compositions to movie files.

Predecessor of the avant algorithm (archival purposes only)

Source ▾

main ▾

d55e4c3 ▾

Full commit



```

175
176     let intrPrint;
177     const intr = () => {
178         intrPrint =
179             "\nDOMINION ALGORITHM\n" +
180             "CODED BY A PIXELATED POINT OF VIEW\n" +
181             "BUILD 1.2.1\n\n" +
182             "REPORT GENERATED AT:\n" +
183             date.toDateString().toLocaleUpperCase() +
184             " - " +
185             date.toLocaleTimeString().toLocaleUpperCase() +
186             "\n";
187         console.log(
188             "\nDOMINION ALGORITHM\n" +
189             "CODED BY A PIXELATED POINT OF VIEW\n" +
190             "BUILD 1.2.1\n\n" +
191             "REPORT GENERATED AT:\n" +
192             date.toDateString().toLocaleUpperCase() +
193             " - " +
194             date.toLocaleTimeString().toLocaleUpperCase() +
195             "\n"
196         );

```

Figure 2: A code-snippet of the Dominion algorithm from late 2022, highlighting its emphasis on rapid development and its prototypical nature⁹

Technically speaking, Avant allows for the generation of practically unlimited variations of potential experimental Quake III machinima films with a few mouse clicks and inputs. While the algorithm's core functionality revolves around Quake III machinima, the algorithm contains additional functionality to simply randomize existing footage captured from an unlimited number of games and/or video sources. Even footage from video cameras and phones may be used with the Avant algorithm; no matter what the user wishes to randomize at that point. If it contains a codec that is compatible with the algorithm, Avant will be able to randomize it and output the required files for use in NLE software or even visualize the output.

Avant was named after the highly popular art movement '*avant-garde*', as it outputs raw, randomized and basically unchecked film data for the primary purpose of creating innovative, experimental and unconventional Quake III machinima films. It is not until the actual post-production process that the user is able to see how the generated data is visualized and captured. The films generated with avant focus on

⁹ Veenstra, J.P. (2025). *Dominion Algorithm: Code Snippet* [Image].

design, graphical representation, artistic expression and the idea of challenging traditional norms and practices found within Quake III movie production – so called ‘*avant-machinina*’ or ‘*algorithmic machinima*’, and not necessarily on their contents nor story; thus staying in true ‘avant-garde’ fashion.

An extensive list of additional forms of functionality for the Avant algorithm has been planned in the future, such as the addition of an API containing extensive information regarding Quake III maps, such as thumbnails, spawn points and locations, map boundaries and an auto downloader function to automatically download and extract the required levels; or the implementation of machine learning: allowing the algorithm to embrace different editing styles (through my own machinima work serving as training data), learning from its choices, properly utilizing them and allowing the algorithm to step away from the pseudo-randomized nature of its generated compositions if desired.

Considering the Avant algorithm will most likely take a few more years to complete because of its complex nature and broad level of functionality, it was decided to develop AvantZero: a proof of concept for the Avant algorithm. Avant remains in active development to this day and finds itself in early stages of development (as of July 2025: version 0.2.9), however, all resources will currently be allocated towards the completion of AvantZero before development on Avant will be resumed. The full Avant algorithm, in its current state, contains a few forms of functionality such as a fully tested and working main menu, a work in progress documentation website, a fully working project manager with CRUD functionality, several functions that ensure proper initialization of the algorithm, a number of general utilities, and a few helper functions for the extraction of map-related data and .bsp files from a .pk3 archive.

The AvantZero algorithm serves as a *proof of concept*: proof that the functionality that is envisioned for Avant really can work as intended and can be further expanded upon. While AvantZero receives less functionality, it contains most of the core pieces of functionality that Avant should contain such as randomized compositions and depth of field map generation. Furthermore, it can output and visualize all generated data in PDF and/or CSV format for data analysts or enthusiasts.



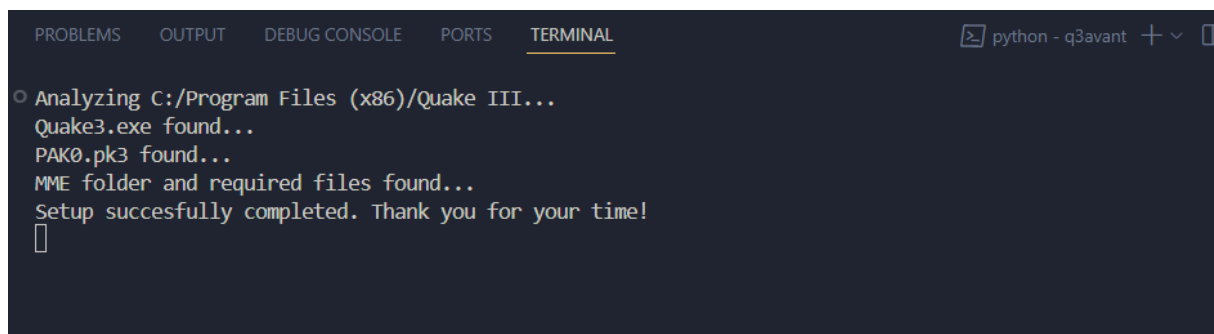
```
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL
PS C:\Users\jordy\Desktop\avant\src\q3avant> python avant.py

  _____
 /  _  \  _  /  _  /  _  /  _  \
 \  _  \  _  /  _  /  _  /  _  \
  \  _  \  _  /  _  /  _  /  _  \
   \  _  \  _  /  _  /  _  /  _  \

Welcome to the Avant algorithm. In order to start using Avant, it needs to know the
location of your Quake III Arena installation.

Please note that you will need the full game (pak0.pk3 is required). The Quake III
de
```

Figure 3: Version 0.2.9 of the Avant algorithm running the first-time setup inside of the command line¹⁰



```
PROBLEMS OUTPUT DEBUG CONSOLE PORTS TERMINAL python - q3avant + v []
Analyzing C:/Program Files (x86)/Quake III...
Quake3.exe found...
PAK0.pk3 found...
MME folder and required files found...
Setup succesfully completed. Thank you for your time!
█
```

Figure 4: Version 0.2.9 of the Avant algorithm has successfully completed its first-time setup¹¹

Once development for AvantZero has been completed, its proof of concept serves a few purposes. First, AvantZero serves as a final submission for CS50X: Harvard’s Introduction to Computer Science and Programming course; thus marking the completion of said course. Secondly, AvantZero will be used for the generation of a few works of ‘avant-machinima’ or ‘algorithmic-machinima’, which serve for both artistic exploration and submission to festivals, galleries, museums and other exhibitions.

¹⁰ Veenstra, J.P. (2025). *Avant 0.2.9 running first-time setup* [Image].

¹¹ Veenstra, J.P. (2025). *Avant 0.2.9 successfully completed first-time setup* [Image].

Considering a personal background as a machinima artist and software developer, AvantZero allows for a highly interesting connection between cinema, code, art and machinima. Lastly, all functionality that has been successfully implemented within AvantZero will be added in the full version of Avant and further expanded upon.

This deep-layered and ever-evolving development approach (Dominion – AvantZero – Avant) illustrates a trajectory of design, technological refinement and deepening of concepts. Each stage serves a purpose in the vision to merge algorithmic processes with create co-authorship in the context of machinima. While limited, Dominion exposed the core functionality but also the core bottlenecks of early automation attempts. It laid the foundation for a more extensible system. Avant, while still in early development, reflects an ambitious take at creating a modular, intelligent, ever-learning and fully integrated piece of software that aligns with efficiency and artistic unpredictability. AvantZero, in this larger context, operates as a technical manifest. It allows for entry into assisted forms of algorithmic machinima and ‘avant-machinima’, opening possibilities for students, researchers and artists to utilize and experiment with both algorithms.

The project, while rooted inside of Quake III Arena, introduces an interesting topic of conversation revolving around how experimental filmmaking can evolve through co-creation, partial automation, machine-based assistance and randomized activity. In this model, the algorithm is not merely a tool but a collaborator as the algorithm outputs decisions concerning the visual aspect of the film, while the filmmaker will remain in control of the auditory scope. Even so, the filmmaker is and will always be able to make any kind of adjustments to the generated sequences: thus, staying in full control of their creative process while always having the possibility to opt in new, exciting forms of hybrid filmmaking thanks to the highly modular philosophy of both the AvantZero and Avant algorithms.

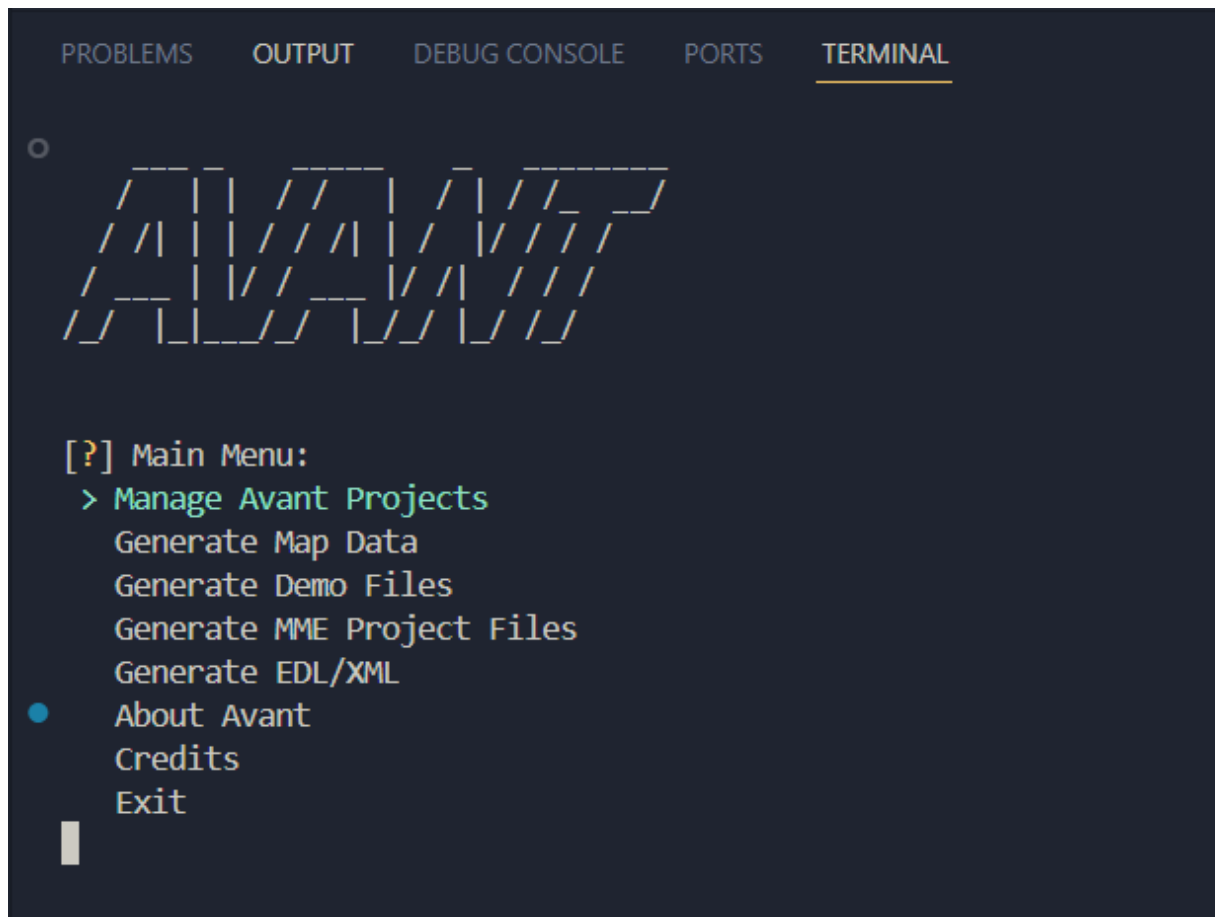


Figure 5: Main menu of the Avant algorithm (Version 0.2.9)¹²

¹² Veenstra, J.P. (2025). Main menu of the Avant algorithm. Version 0.2.9 [Image].

Chapter 2 – Technology Stack

Both the AvantZero and Avant algorithms are entirely developed with Python and utilize several internal and external (PyPi-based) libraries. The choice for Python was motivated through a combination of practical, technical and educational reasons.

First, Python’s combination of clear syntax and object-oriented programming facilitates speedy development while maintaining clarity and modularity across the codebase. This is crucial given the layered and class-based structure of AvantZero, which spans across multiple files and dynamically linked components.

Secondly, Python offers an extensive ecosystem of ready-to-use libraries for data analysis, visualization, depth estimation and further means of automation; resources that significantly contribute to the development and refinement of AvantZero’s algorithmic components and workflows. A few great examples of ready-to-use libraries are: *matplotlib* (for visualizing and plotting data), *reportlab* (for generating PDF files containing text and/or images) and *DepthAnything* (Scientific Python-based Depth of Field estimation model that works with both image and video).

Equally essential to the project was the implementation of a Graphical User Interface. Although frameworks such as Microsoft .NET MAUI (using the C# programming language) or Electron (JavaScript) are popular choices for cross-platform GUI-based development, Python’s native tkinter library proved to be a pragmatic and effective solution. Its simplicity and direct integration with the rest of the codebase and ease of customization made it ideal for any of the algorithmic requirements. Importantly, tkinter allowed for Python to be maintained as the single language across all layers of the project; from logic to interface, reducing overhead and improving maintainability. Packaging tools such as PyInstaller were used to convert the project into standalone executables for Windows and MacOS operating systems, making deployment seamless and providing users with direct access to the algorithm with little to none setup required.

Lastly, the choice of Python as the sole programming language for the AvantZero algorithm was also personally motivated due to broad familiarity with its ecosystem, syntax, packages and was also pedagogically motivated because of a personal educational background as a Web Development and Python lecturer. By creating the

AvantZero algorithm solely within Python, it could serve an additional purpose as a dual-point of inspiration and (case-study) reference for students over the course of the two marking periods during which they would be following Python lectures. Many of the fundamental aspects found within AvantZero, such as the use of functions, loops, objects, classes, internal libraries (such as *os*¹³, *sys*¹⁴ and *random*¹⁵) and external libraries (such as *matplotlib*¹⁶, *requests*¹⁷ and *fastAPI*¹⁸) are core programming concepts found within the Python lectures: allowing for practical examples of how said concepts and libraries could be integrated into real-life software projects.

Future functional additions centering around machine-learning and artificial intelligence during the later stage of Avant's development cycle with libraries such as *TensorFlow*¹⁹ and *PyTorch*²⁰ would also benefit greatly from use with the Python programming language as Python is a dominant language within the *machine-learning* (ML) and *Artificial Intelligence* (AI) domain; making the language an ideal choice for such future expansions. The ability to pretty much natively integrate ML models into an existing Python codebase without major restructuring ensures that future releases of Avant can continue to expand while remaining technologically coherent.

Lastly, we'll briefly touch upon some of the internal and external Python libraries used during the development cycle and further explain their place within the software's hierarchy. Considering that a development cycle consists of many twists and turns both from a functional and library-based perspective, only the most important libraries that are sure to be embedded within the AvantZero algorithm will be further introduced.

¹³ Python Software Foundation. (n.d.). *os* – Miscellaneous operating system interfaces. Python 3. Retrieved July 19, 2025, from <https://docs.python.org/3/library/os.html>

¹⁴ Python Software Foundation. (n.d.). *sys* – System-specific parameters and functions. Python 3. Retrieved July 19, 2025, from <https://docs.python.org/3/library/sys.html>

¹⁵ Python Software Foundation. (n.d.). *random* – Generate pseudo-random numbers. Python 3. Retrieved July 19, 2025, from <https://docs.python.org/3/library/random.html>

¹⁶ Matplotlib Development Team (n.d.). *Matplotlib* – Visualization with Python. Matplotlib. Retrieved July 19, 2025, from <https://matplotlib.org/>

¹⁷ Python Software Foundation. (n.d.). *Requests: HTTP for Humans*. Python 3. Retrieved July 19, 2025, from <https://requests.readthedocs.io/en/latest/>

¹⁸ Ramirez, S. (n.d.). *FastAPI*. FastAPI. Retrieved July 19, 2025, from <https://fastapi.tiangolo.com/>

¹⁹ Tensorflow (n.d.). *Tensorflow*. Tensorflow. Retrieved July 19, 2025, from <https://www.tensorflow.org/>

²⁰ PyTorch Foundation (n.d.). *PyTorch*. PyTorch. Retrieved July 19, 2025, from <https://pytorch.org/>

We'll briefly touch upon the following internal- and external Python libraries:

- OS
- Sys
- DepthAnything
- Tkinter

OS

The built-in Python library *OS* is mandatory for AvantZero's development cycle. It allows us to communicate with the operating system through several ways. For instance, we're able to list all the files and folders within a certain directory, create the AvantZero folder structure, automatically create the proper folder structure for any of the projects made with the algorithm and save, adjust or delete any of the created files. Thanks to the *OS* library, any form of communication between Python, the operating system and its files or folders becomes a streamlined process.

Sys

Another built-in Python library which contains a few features that allow us to communicate with the operating system from a more technical point of view. However, it must be noted that *sys* and *os* are not the same. The *sys* library features functionality with which we can directly terminate the application and free the allocated memory, start the application with additional arguments from the command line, and present more technical information regarding the system AvantZero is being executed on, such as hashes, hex codes and information regarding the operating system we've used to run the algorithm.

DepthAnythingV2

A monocular depth estimation model written with Python and PyTorch, using machine-learning to estimate depth of field inside images and video's. Recently, a new groundbreaking model has been released called *VideoDepthAnything*²¹, which provides the user with consistent depth of field map generation for longer types of video content (5 minutes or longer).

The model has been extensively documented by several means: through academic writing²², GitHub²³ and its official website²⁴. Furthermore, the developers have provided a demo version of DepthAnythingV2 on HuggingFace²⁵; allowing users to experiment with the model directly.



*Figure 6: Picture of roses in a vase, presented within a depth-of-field map.
Captured with the Depth Anything V2 demo found on Hugging Face²⁶*

²¹ DepthAnything (n.d.) *Video Depth Anything: Consistent Depth Estimation for Super-Long Video's*. GitHub Pages. Retrieved July 22, 2025, from <https://videodepthanything.github.io/>

²² Yang, L., Kang, B., Huang, Z., Zhao, Z., Xu, X., Feng, J., & Zhao, H. (2024). Depth Anything V2. In *Proceedings of NeurIPS 2024*. arXiv. Retrieved July 22, 2025, from <https://arxiv.org/abs/2406.09414>

²³ DepthAnything (2025). *Depth-Anything-V2*. [GitHub Repository]. GitHub. Retrieved July 22, 2025, from <https://github.com/DepthAnything/Depth-Anything-V2>

²⁴ DepthAnything (n.d.). *Depth Anything V2 project page*. GitHub Pages. Retrieved July 22, 2025, from <https://depth-anything-v2.github.io/>

²⁵ Hugging Face (n.d.). *Depth Anything V2 Demo*. Hugging Face. Retrieved July 22, 2025, from <https://huggingface.co/spaces/depth-anything/Depth-Anything-V2>

²⁶ Veenstra, J.P. (2025). *Using Depth Anything V2 on Hugging Face* [Image].

Tkinter

Tkinter is a built-in Python library used to create Graphical User Interfaces (GUIs) for applications. Since Python is often executed and interacted with through the command line, Tkinter enables developers to build engaging and clear visual interfaces for software programs. The library offers numerous tools for creating splash screens, buttons, selection menus, loading bars, and customizing the design of these elements—allowing AvantZero to stand out with a polished user experience.

Because everything runs within the Python ecosystem, using Tkinter allows the application to scale gracefully without relying on external interfaces or additional libraries.

Chapter 3 – Goals and Objectives

The main goal for the AvantZero algorithm is to fully automate various tasks found within Quake III machinima (post)production pipelines; tasks that would otherwise need to be conducted manually. Keeping the algorithm's artistic and technical origins in mind, it aims to achieve the following creative and technical objectives:

Generation of unique, pseudo-randomized and experimental video sequences

AvantZero enables users to generate pseudo-randomized experimental video sequences by reorganizing pre-existing footage in an algorithmic way that defies repetition. Each composition generated by the algorithm is to most extent unique, even when using the same source material. This process of automatically generating new content reflects the algorithms underlying commitment to avant-garde principles, such as spontaneity, variation and non-linearity instead of embracing traditional and narrative-based ways of filmmaking.

Rather than following predetermined editing patterns, AvantZero outputs randomized post-production data (in formats such as XML and EDL) that can be imported directly into a non-linear editing (NLE) software suite. The algorithm acts as a generative and creative partner, proposing unexpected visuals and rhythms that challenge conventional filmmaking logic.

The algorithm aligns with experimental filmmaking traditions that aim for abstraction and viewer interpretation. By automating the visual structure through the algorithm and leaving the auditory domain to the user (such as sound design, audioscaping, music and/or dialogue), AvantZero introduces a highly interesting form of collaboration between human and code. While this approach of algorithmic machinima-making may seem closed-circuit in a sense, it must be noted that the filmmaker will always remain in control of their creative choices.

The EDL and XML files are not locked in any shape or form: allowing the user to change, improve upon or revert changes conducted by the algorithm. Any output generated by AvantZero may be fully accepted by the user, rejected, or used as a starting point for further improvements. Therefore, avant-machinima, or

algorithmic-machinima is an exciting and continuously evolving body of work where no two outputs are identical.

Automating the assembly process of diverse montages

In virtually every filmmaking workflow, the act of assembling and trimming raw footage remains a foundational process. Some might find it a repetitive process. This is no different in machinima. Machinima artists are often tasked with selecting, sequencing and refining a vast array of gameplay clips, and sit through large quantities of video frames. To address the elephant in the room: AvantZero is not seeking to replace the manual editing process, neither will it propose or promote automation as a substitute for artistic expression. On the contrary, the manual curation of footage should be valued and preserved as rhythms, contrasts and visual leitmotifs appear organically throughout the process. It is within this very hands-on manipulation that a film's unique sense of style begins to take shape.

Rather than suppressing this process, AvantZero serves as a complementary pillar. It is a system of visual experimentation designed to challenge traditional conventions by primarily emphasizing focus on experimentation and randomized footage selection. It embraces the idea of randomizing its footage selection, and adoption of uncertainty and unpredictability. It avoids the familiar pacing found in traditional cinematic convention. So much more so of uncertainty and unpredictability is found in the full version of AvantZero (the Avant algorithm), in which the user is only able to witness their final result once they import the XML/EDL file into the NLE for the first time and load up the generated sequence.

AvantZero introduces a new audiovisual syntax that privileges experimentation over story; variation over repetition. It supports an exploratory mindset, where the algorithm serves as a collaborative tool inside of the editing suite. The user can adjust the proposed edits to their liking at any given time, or, if desired, immediately accept the presented composition. The algorithm serves as a provocative starting point for further refinement, disruption, distortion (by utilizing the Practice of Distortion Framework²⁷), reinterpretation and cinematic expression. AvantZero both embraces

²⁷ See: Bittanti, M., & Veenstra, J.P. (in press). *Los Santos Plays Itself*. Mimesis International.

and restructures the creative loop between intuition, interpretation, tradition and experimentation.

Broadening the roles of the User (Post-Editor & Interpreter)

By default, during most film post-production processes, the user acts as the author. While using the AvantZero algorithm, the user gains an additional role: that of *interpreter*. The editorial agency is initially shifted from the user towards the algorithm, since it is the algorithm that presents the user with a preliminary and randomized sequence. While doing so, the user's ability to intervene, reject and/or reinterpret remains preserved. By shifting the roles, the process of editing transforms into a form of dialogue between human and machine; between suggestion, interpretation and creative desire.

Multi-Format Compatibility and Integrated Depthmap Generation

AvantZero was designed with flexibility in mind. Thanks to a multitude of external software libraries implemented within the algorithm, such as the DepthAnythingV2 model, it accepts both image sequences and traditional video formats, such as, but not limited to: JPG, PNG, TIFF, DPX and H264, Apple ProRes, MXF, HEVC or AVI files. The algorithm's wide range of compatibility ensures that artists can work across a broader spectrum of visual material.

While the codecs above exemplify several codecs that will enjoy support from the first initial version, it remains a possibility that initial support will be extended with additional codecs during later stages of development; considering that minor updates will still be considered during AvantZero's significantly shorter development cycle (due to its proof of concept status).

AvantZero's multi-format file compatibility allows for deeper integration into diverse post-production pipelines and, in tandem with its XML and EDL generation capabilities, for direct compatibility with several NLE's as well; such as Adobe Premiere Pro, Blackmagic Da Vinci Resolve and/or Avid Media Composer.

In addition to its compositional functionality, the algorithm contains the capability to generate frame-by-frame depth of field image-streams alongside (or in a standalone

type of fashion) its randomized video output thanks to the groundbreaking *DepthAnythingV2* model. These depth maps provide valuable metadata that can be used in a variety of technical workflows (such as 3D compositing, generating manual depth of field masks) or for further experimental and artistic intent.²⁸

Organized Data and Clear Visualizations in a multi-format fashion

While the core focus of AvantZero revolves around generating randomized experimental data and video sequences, the software is also equipped with tools for organizing and outputting generated and randomized data in a structured and accessible format. Generated datasets are automatically compiled into clear PDF reports, and exported as CSV files for further processing or analysis in conventional environments such as vanilla Python, conda or Jupyter Notebooks.

In addition, AvantZero supports the automatic creation of visualizations: such as line graphs, bar charts, and scatter plots; using libraries like *matplotlib*. Generated plots are exported both as part of the PDF reports and as individual image files (e.g., PNG), allowing for use in presentations, publications, or external dashboards.

All organizational and visualization functions are tightly bound to each individual project. This means they can always be recalled, regenerated, or updated at any stage of the workflow, ensuring consistency throughout the experimental process.

²⁸ A Pixelated Point of View (2023). Quake III Arena: Experimental Depth Map Use [Video]. YouTube. Retrieved July 22, 2025, from <https://www.youtube.com/watch?v=7cu76qMIWIQ>

Chapter 4: Data Structures & System Workflows

AvantZero will utilize the Object-Oriented Programming paradigm during its development cycle, which lays emphasis on the use of classes and instances (named objects) to create extensive and complex relationships between the different cores found within the software. In AvantZero's design, each larger encompassing aspect of the software will be considered a '*core*', or in OOP-terms: a class.

Relationships between cores are visualized with the aid of UML (Unified Modeling Language): a set of standardized diagrams used to depict the structure and interactions within object-oriented systems. Several times throughout this chapter, we're referring to what is called a *sequence diagram*. A sequence diagram is a visual representation of the relationships and connections between objects and how they interact with each other over time.

In the case of the AvantZero algorithm, the diagram depicts all cores found within the software, how each of these parts consecutively interact with one another and visualizes the data exchanged between each of the various cores to reach a specific type of output. Considering each of the core's interconnect with one another several given times during the data generation process, the diagram required some dilution to visualize the phenomena and processes at play in the most clear and concise way possible. After introducing the diagram, all cores will be briefly further elaborated upon, with the chapter's conclusion centring around a brief and concise explanation of the processes that are at play within the proposed sequence diagram.

The diagram is divided into two distinct areas, referred to as '*Boot*' and '*Main*'. All functionalities found within the boot section of the diagram is linked to the initialization and booting process of the algorithm. The main section centres around all other form of functionality, such as the project manager (PROJM), data analysis generator (DANLY), depth generator (DEPGEN) and the main system (SYS); which concerns itself with all higher-level functionality and connectivity between all cores found within the algorithm.

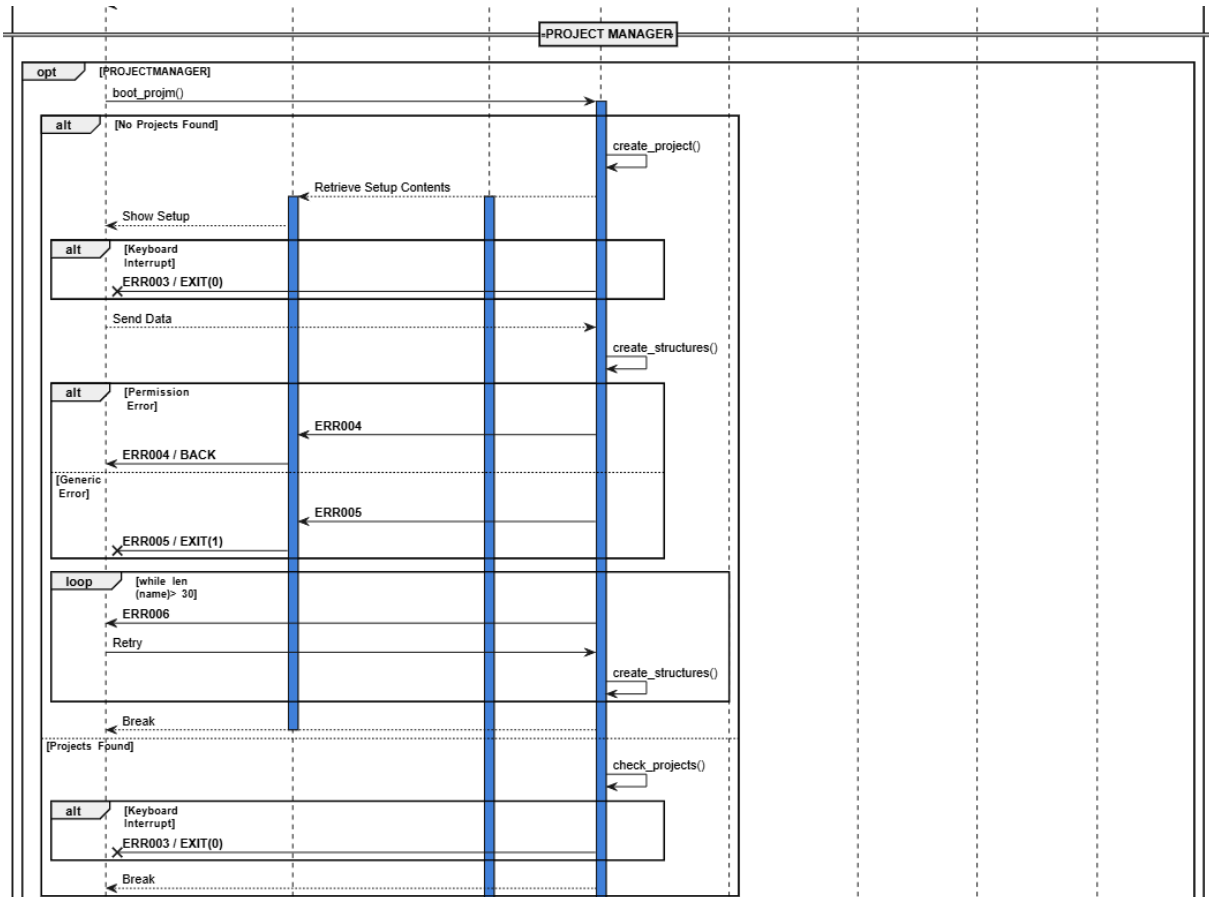


Figure 7: Partial representation of the AvantZero Sequence Diagram: visualizing the initial checks conducted by the project manager after boot²⁹

Considering the large size of the diagram, each of the different sections found within the diagram will be represented, introduced and elaborated upon individually: keeping digestibility and conciseness in mind. For the full diagram, please refer to the AvantZero GitHub Repository.³⁰ The source-code for the sequence diagram, and visualizations of the entire diagram in various formats (SVG, PNG and PDF can be found within the *docs* folder.

The diagram was created inside Visual Studio Code with the aid of the *PlantUML*³¹ library in a bit over four-hundred lines of code. PlantUML is a Java-based tool allowing for the streamlined creation of various UML diagrams, such as sequence diagrams and activity diagrams.

²⁹ Veenstra, J.P. (2025). *Partial Representation of AvantZero Sequence Diagram* [Image].

³⁰ Veenstra, J.P. (2025). *AvantZero*. [GitHub Repository]. GitHub. Retrieved July 28, 2025, from <https://github.com/jiyorude/avantzero>

³¹ PlantUML (n.d.). *Quick Start Guide*. PlantUML. Retrieved July 28, 2025, from <https://plantuml.com/starting>

Together with the PlantUML extension found within Visual Studio Code, users can create clear, neat and customized diagrams with code-based components inside *.puml* files, easily visualize and/or export them to tangible files for later use.

AREA 1 - BOOT

1.1 – Initial Boot

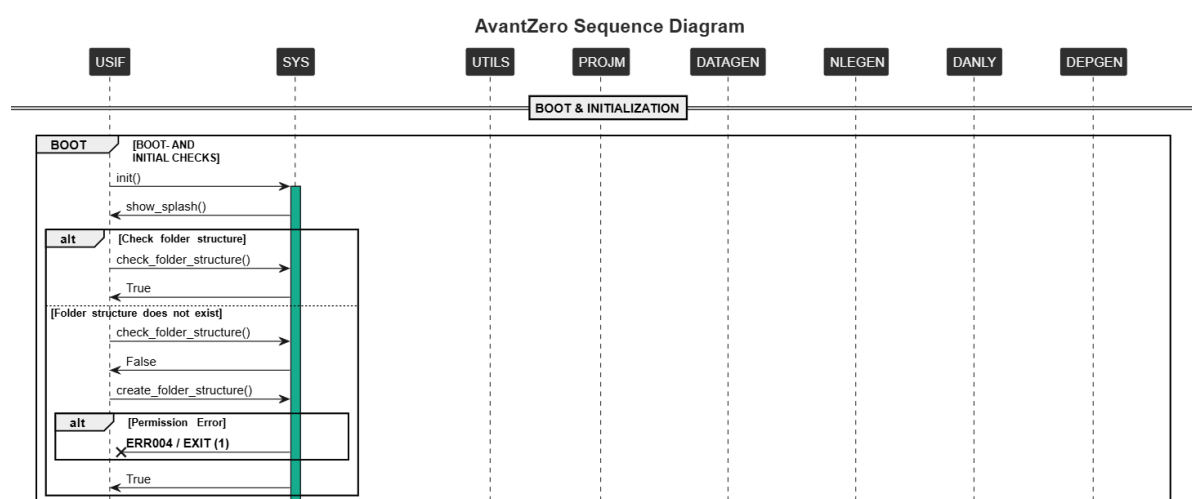


Figure 8: AvantZero Sequence Diagram: Visual representation of the initial boot functions³²

Figure 8 above contains a partial representation of the first distinct area found within the AvantZero algorithm referred to as *BOOT*. The functionality described here mainly revolves around the boot and initialization process of the algorithm. A few examples include creating the splash screen as a form of visual indicator that the algorithm is booting, conducting initial checks of all required Python files, verifying if the AvantZero folder structure exists and has remained in place since the last boot, and whether all required Python libraries have been successfully installed and rightfully called upon. It is not until the working of all vital components has been

³² Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of Initial Boot Functions* [Image].

successfully verified that initial control is given back to the user and the main menu is presented.

Before delving into the specific connections illustrated in the diagram, a quick introduction is in order of all the various components found within the diagram. Each of the core modules within the software is depicted inside the black and white boxes located at the top and bottom of the diagram. These modules, or ‘cores’ are arranged along the Y-axis, with their vertical alignment marked by dotted lines to indicate their position inside the sequence flow. In UML sequence diagrams, such conditional logic is represented through *alt* blocks, which visually represent the branching logic found within if-else statements. The ‘alt’ section represents the ‘if’ path, while the optional ‘else’ section outlines the alternative possible outcome.

Arrows between these cores represent interactions or function calls and are often accompanied by a label that either names the invoked function or briefly describes the performed operation. In some cases, the diagram presents scenarios where different outcomes are possible depending on certain situations. In programming, such situations are typically handled through ‘if-else’ statements. These structures evaluate a condition: if the condition is true, the code within the ‘if’ block is executed; otherwise, the ‘else’ block is executed. Whenever a connection is made between cores, one of these cores becomes ‘activated’ as it is called upon. Colorful blocks (such as the teal block seen in the diagram above) serve as a visual representation of a core’s active state.

In short, the initialization process, as depicted within the diagram, goes as follows:

1. The algorithm is executed, either through an executable file or a CLI-based command.
2. The **init** function is called (found within the system or *SYS* core), telling the system to start the actual initialization process.

3. The system core responds to the request by providing the User Interface (*USIF*) with the splash screen; a visual indication for the user that AvantZero is actively initializing all its dependencies and has indeed received the boot request.
4. Once the splash screen has been presented, the algorithm enters its first alt-block as it needs to conduct several checks ‘under the hood’ of the software. One of these checks encompasses whether the AvantZero folder structure currently exists or still contains all the required folders and/or if files needed by the algorithm for project management- and data generation purposes are still found in all the right places. The splash screen activates the function *check_folder_structure()* and tells the system library to conduct the check. If everything was found to be in order, the function will return the value *True*; indicating that the check was completed successfully. If the check were to fail, for example if the algorithm was booted for the very first time, the check will return *False* and subsequently boot a second (helper) function that creates the required structure. Afterwards, the check is conducted a second time to verify if all files and folders have now been put in their respective places.
5. At any given time, it may be possible for the algorithm to experience OS-related issues, such as read-write limitations within certain folders. If this were to be the case, the algorithm will show error code 004: a so-called permission error and prematurely terminate the booting process considering the scope of the error would require manual intervention by the user and a reboot of the algorithm to be resolved successfully.

1.2 – Dependency- and Package Checks

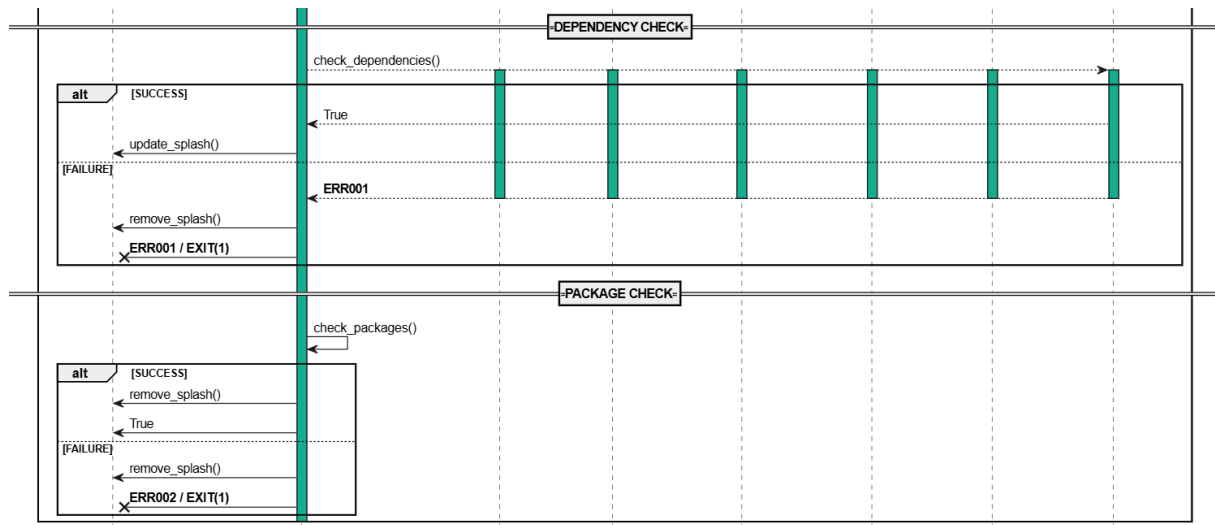


Figure 9: AvantZero Sequence Diagram: Visual representation of the dependency and package check functions³³

Upon completion of the first check, the folder structure check, two additional checks are conducted while the splash image is visible on screen. These are referred to as the *Dependency Check* and *Package Check*. The dependency check is automatically and subsequently called after the completion of the folder structure check.

1. The dependency check conducts a brief investigation into whether all cores exist and can be accessed by the software. Considering the open-source nature of the software, the check will not conduct any investigations regarding the functions and/or the code found within the cores considering the repository may be forked, adjusted, re-released and further expanded upon by third parties once the algorithm reaches the end of its development cycle.
2. If the check fails, the function returns *false* and triggers Error Code 001 through a pop-up screen; removing the splash image from the screen in the process. As the algorithm is unable to proceed with the initialization process,

³³ Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Dependency and Package Check Functions*. [Image].

the process will simply be terminated as the algorithm will need file/code-based adjustments and at least another restart to successfully resolve the issue.

3. If the dependencies were successfully checked, the splash image text and the loading bar will be updated to notify the user.
4. The third and final check consists of investigating the so-called package import statements. These are found throughout almost all Python files found within the repository and are used to import functionality from internal Python packages and external PyPi packages alike. Each of these packages, along with their import statements, are checked upon every single boot to verify whether all internal packages can be accessed and if the required external packages have been properly installed by pip (Python's package manager), can be found by Python and, in fact, are able to provide the required functionality to the algorithm. If any of the packages were to fail, error code 002 is supplied to the user: indicating there's something wrong with one of the packages. The algorithm will be unable to continue with the initialization process as it depends on these packages to guarantee full functionality. Furthermore, action is required by the user as the failed packages will need to be reinstalled or checked. If this situation occurs, the user will be informed of the error through a pop-up screen, the splash image will be removed from the screen and the initialization process is subsequently terminated. The function is initiated from within the system core and the result will be returned to that same system core, hence why the function originates and points towards the same core.
5. If all three checks have been successfully passed, the loading bar will be set to 100% and the splash image will be removed. A different function (`boot_mame()` – **boot main menu**) will be called to boot the main menu of the algorithm and present the menu inside of the User Interface. This function will only be called if all three checks have returned to a true value.

AREA 2 - MAIN

2.1 – Main Menu

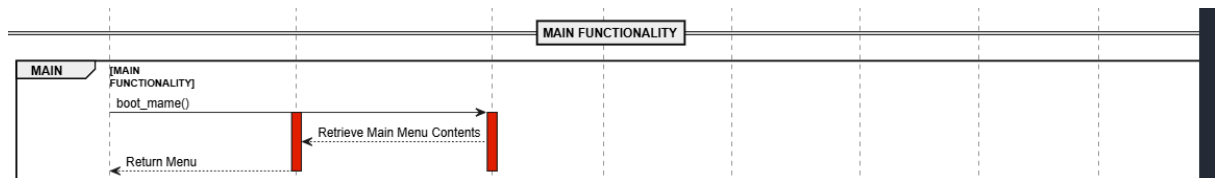


Figure 10: AvantZero Sequence Diagram: Visual representation of the `boot_mame()` function³⁴

With the initialization and booting process complete, the user moves on to the second and most extensive area of the algorithm, referred to as *MAIN*. It is at this point that, to at least a basic extent – without taking any other possible errors into consideration, we should be able to provide functionality to the user and aid the user throughout their entire journey. The user is now given full control of the algorithm and can select any function pretty much at will. In UML-terms, the user is given several options. They could boot into the project manager, generate depth of field maps, check the credits section or gracefully exit out of AvantZero.

Before each of the different options is introduced, let's take a brief look at the function that retrieves the main menu:

1. With all three initial checks completed, the boot main menu function is called. The function connects directly to the system and utilities cores (for additional helper functions).
2. The helper functions found within the utilities core are returned to the system core. This includes all text shown on screen (there will be a one centralized Python file containing all lines of text found in the algorithm to provide easy reference and adjustments later).

³⁴ Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the `boot_mame()` function.* [Image].

3. The system core returns the contents to the User Interface and the main menu is visually presented to the user on screen. The user is now able to navigate through the menu by either clicking on their choice function or by using keypresses.

2.2 – Project Manager

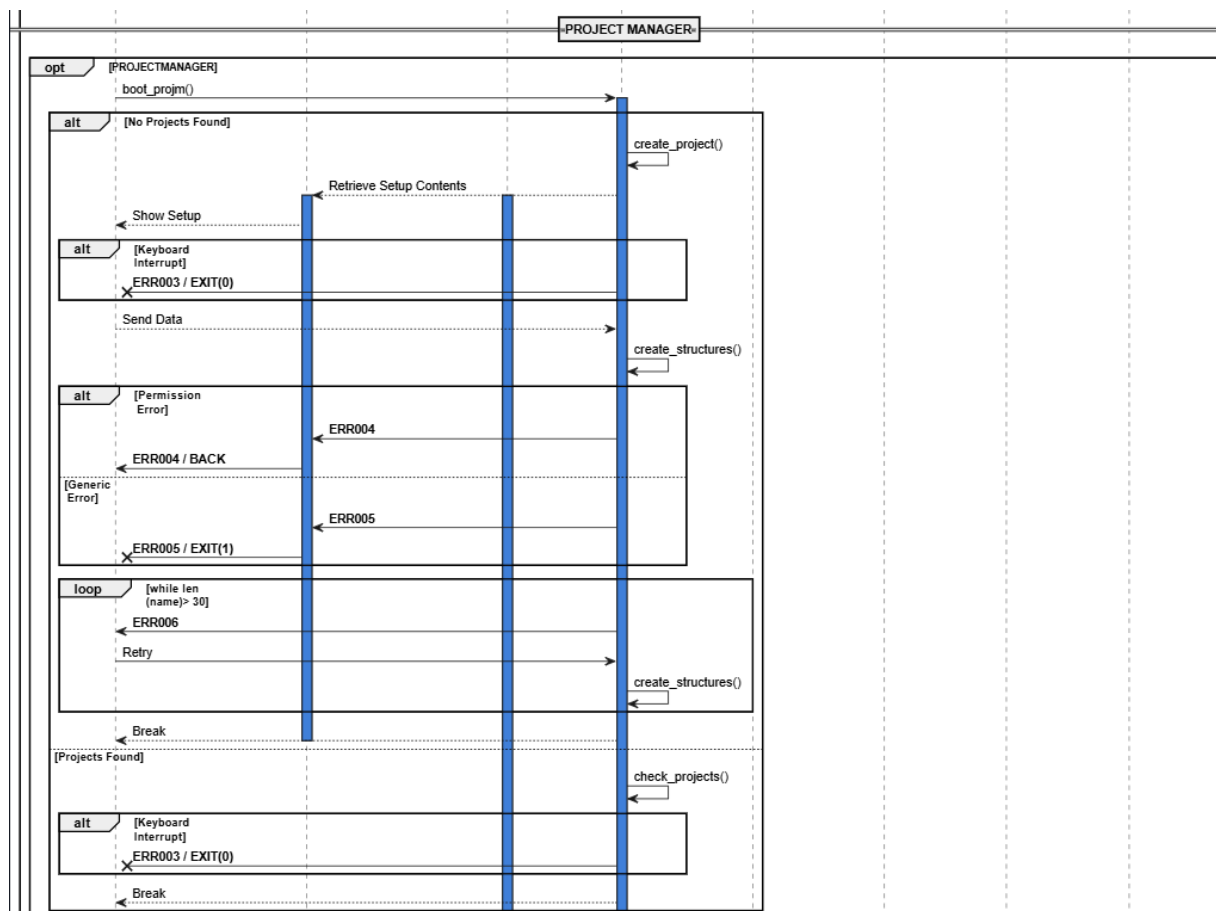


Figure 11: AvantZero Sequence Diagram: Visual Representation of the Project Manager Boot Process ³⁵

The first, main form of functionality that AvantZero offers is referred to as the *project manager*. The project manager allows the user to execute several CRUD-based tasks regarding Avant-projects, which serve as the main source of information for the algorithm. The user can view projects and any of the generated data or project

³⁵ Veenstra, J.P. (2025). AvantZero Sequence Diagram: Visual Representation of the Project Manager Boot Process. [Image].

metadata, edit projects, create new projects or delete any of the projects created. The algorithm itself uses project folders to store generated data and usage information, such as how many versions of a project exist.

Before the user is given control of the project manager, it conducts a preliminary check to see whether avant projects exist in the Avant folder. Naturally, two situations might occur. Either the Avant folder is completely empty and doesn't contain a single avant project, or there have been projects created in the past.

1. When the project manager is selected in the main menu, it executes a function within the PROJMAN or project manager core named *boot_projm()*, which conducts whether existing projects exist inside of the Avant folder. If this is the case, then, from a basic perspective and again not taking any complex errors into account, the user should be able to utilize most of the functionality found in this *option* and the function will simply return true and break out of its loop. It is possible, at any given time after the initial boot process has been completed, that the user might conduct a so-called 'Keyboard Interrupt': a fail-safe built in AvantZero utilizing basic Python terminology (often conducted with the CTRL + C keys) that automatically and instantaneously terminates the algorithm if it were to be called. Error Code 003 is reserved for Keyboard Interrupts, and will, if called, provide the user with visual feedback that the algorithm will be prematurely terminated due to a Keyboard Interrupt.
2. Several options might come to the table if the check returns empty-handed, which in our case results in a false return statement. Avant will require at least one project whenever the project manager is booted to ensure full compatibility. In this case, the algorithm will call upon the *create_project()* function found within the project manager and guide the user through a setup-based wizard. It will ask the user for the project's name, framerate, summary, video resolution and other required (technical) information for future references. This data is used for data generation and EDL/XML creation, to name a few examples. Note how the length of the film is currently not included. Considering the user might want to create different versions of their footage consisting of shorter and longer runtimes, this peculiar insight will be further expanded upon later.

3. Once AvantZero has received all input data, it will automatically create the data structures required by sending all the data back to the project manager's core. Once the (sub)folders are in place, *the boot_projm()* function will be called again: now returning a true value and breaking out of its loop.
4. A few issues might arise during project creation, such as permission issues imposed by the operating system. If this were to be the case, AvantZero returns an error code 004 and returns the user back to the main menu as technically speaking, the user would be able to try again after configuring several settings. For now, to ensure full compatibility and maintain clarity, users are limited to a maximum of 30 characters in their project names as date and timestamps are concatenated automatically in the project folder. Whenever this occurs, the algorithm will keep prompting until the user provides a name less than 30 characters (or whenever the user manually exits out of the algorithm, of course – whichever happens first) and only then will proceed by creating all the required data structures. All other errors are 'caught' in a singular error, referred to as an *Exception* and referred to as Error Code 005, which will prompt both the termination of the algorithm and a pop-up window for the user for quick reference.

2.2.1 – Create Project

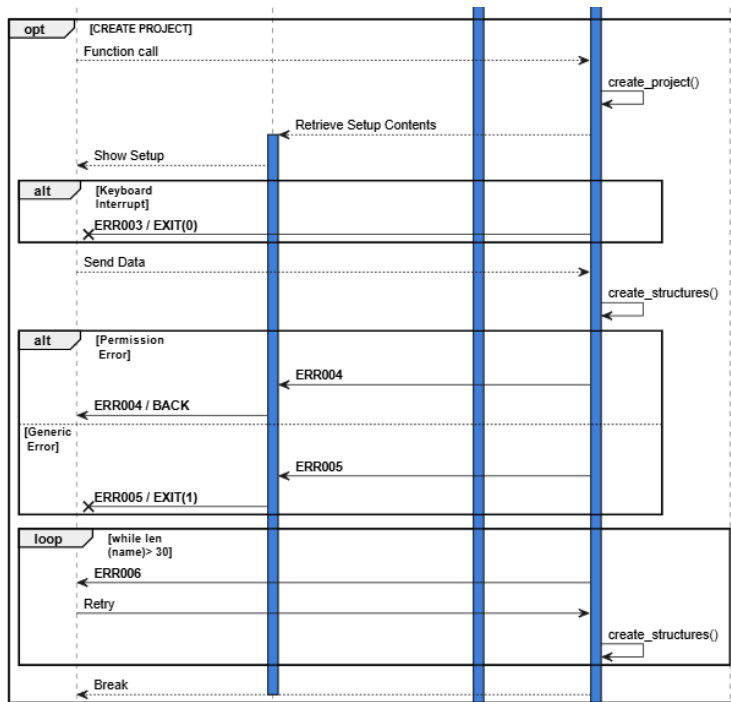


Figure 12: AvantZero Sequence Diagram: Visual Representation on creating an Avant Project ³⁶

The user can initiate a new project at any point via the Project Manager menu, utilizing the integrated project creation tool. As illustrated in Figure 12. There is a noticeable resemblance to the layout and flow presented in Figure 11. This similarity is intentional; the core logic behind the project creation process remains consistent, whether a project is initialized manually by the user or automatically during the execution of the *boot_projm()* function during startup. Figure 12 is intended to serve as a visual overview of the project creation process and its interface, rather than repeating a detailed explanation of the underlying functionality as this has been conducted within the previous chapter.

³⁶ Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation on creating an Avant Project*. [Image].

2.2.2 – View Project

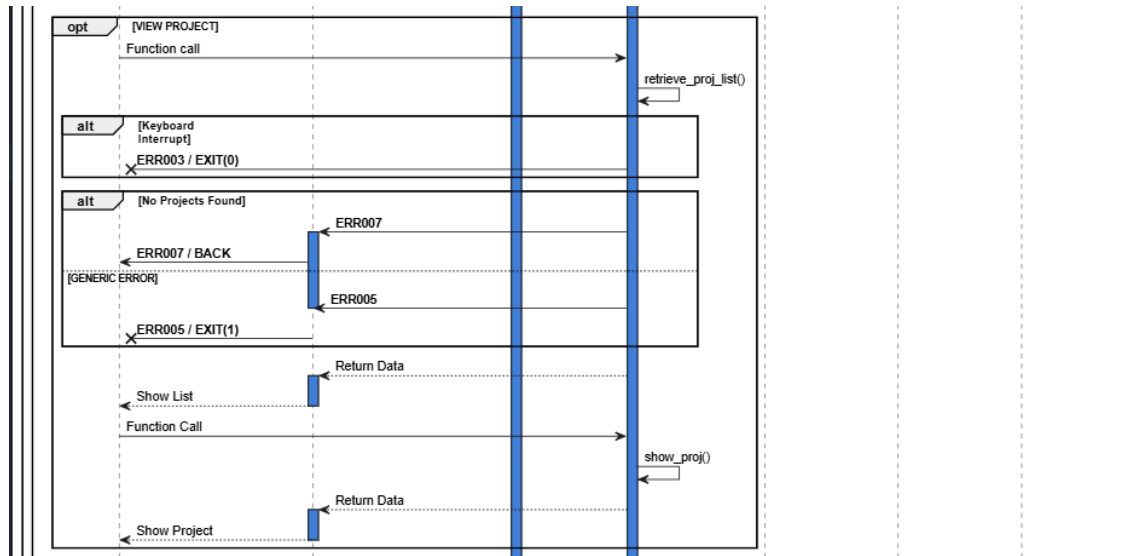


Figure 14: AvantZero Sequence Diagram: Visual Representation on viewing an Avant Project ³⁷

Users can access all previously created projects within the ‘View Projects’ menu. The menu provides the user with a clear overview of different key parameters found within each project, such as the name of the project, a description, and timetables indicating when data was generated or exported, allowing users to easily distinguish between different data batches. Additionally, the menu offers a summary of the input files found within the project folder, detailing aspects such as length, format and the codecs used. Intentionally, the menu provides the user with valuable insights into the compatibility of each file: rapidly and concisely determining whether the files can be successfully used for future data generation or export operations within the algorithm.

1. The View Project menu is selected; sending a function call towards the project manager core. It will retrieve a list of projects found within the Avant folder and return it to the system core. The system core visually presents the information to the User Interface.

³⁷ Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation on viewing an Avant Project*. [Image].

2. The user scrolls through the User Interface, selecting any project they would like to view. The user confirms their selection, which subsequently sends another function request towards the project manager. The `show_proj()` function retrieves the requested project data core and is printed to the User Interface thanks to the system core.
3. At any given time, the user is free to conduct a complete manual termination of the algorithm by pressing CTRL + C; a so-called Keyboard Interrupt (Error Code 003). The user is greeted with a pop-up screen: confirming that the algorithm will be shut down.
4. If the algorithm encounters any kind of generic error (or Exception) that would prevent the algorithm from successfully completing its task, it will throw an Error Code 005 at the user before automatically shutting down.
5. If the user were to complete the mandatory project creation setup while booting the project manager, continue by deleting their newly created project and open the View Project menu, the algorithm shows a pop-up screen indicating that the project manager is currently empty, before returning them to the previous screen automatically (the mandatory project creation setup would be triggered again if the user would re-enter the project manager from the main menu).

2.2.3 – Modify Project

Avant projects may be modified for several reasons. The setup allows the user to change a few project parameters, such as the framerate (earlier generated data won't be affected - this might become a potential function for the Avant algorithm), project names and project descriptions.

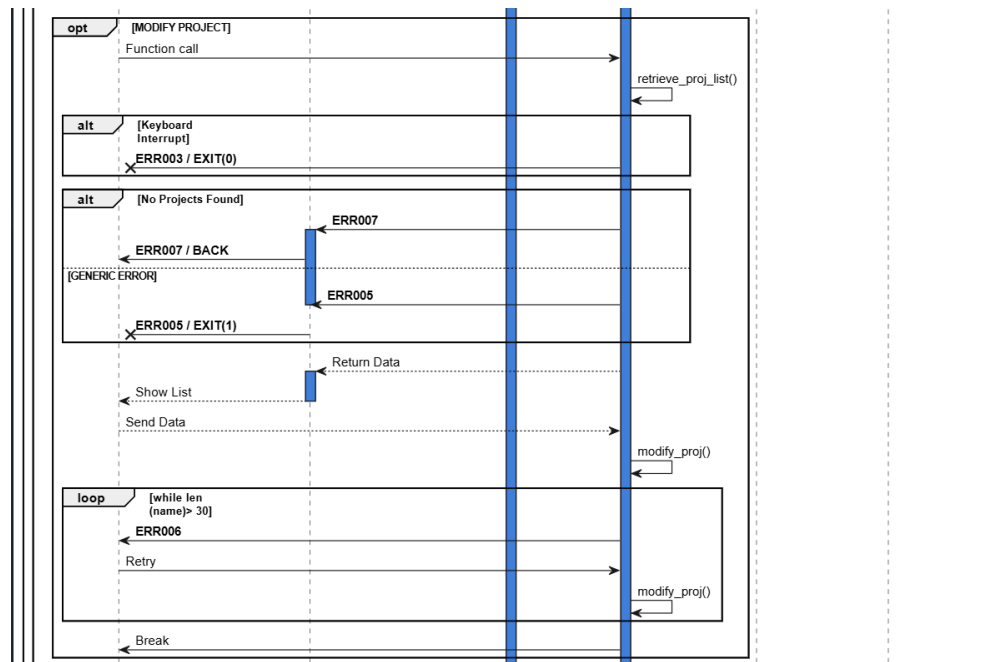


Figure 15: AvantZero Sequence Diagram: Visual Representation on modifying an Avant Project ³⁸

1. The modify project function is called, which subsequently retrieves a list of all currently existing projects, returns their data and presents it to the User Interface.
2. The user selects a project and returns their choice to the *modify_project()* function directly.
3. The user adjusts the data and confirms their inputs. The project is adjusted, the loop breaks, and the user is returned to the project manager.

³⁸ Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation on modifying an Avant Project*. [Image].

2.2.4 – Delete project and return to main menu

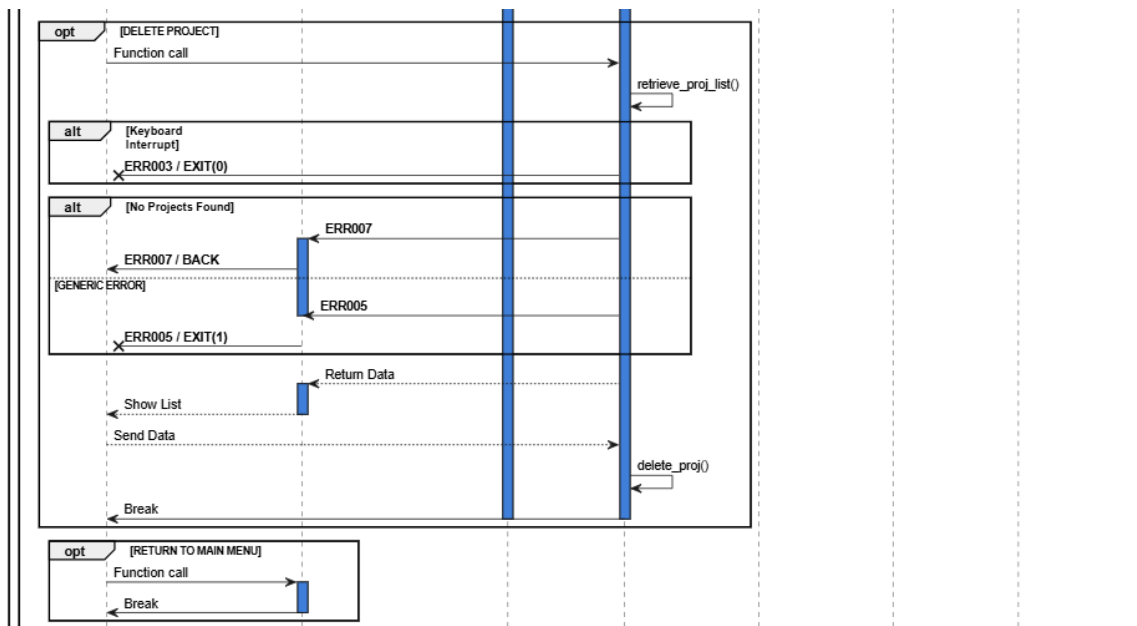


Figure 16: AvantZero Sequence Diagram: Visual Representation on deleting an Avant Project and Returning to the Main Menu ³⁹

The final key functional component within the Project Manager is the ability to delete projects, thereby completing the full CRUD cycle (Create, Read, Update, Delete) essential for managing Avant projects. This functionality allows users to not only create, view and modify projects, but to also remove them in their entirety when they are no longer needed. To enhance user experience and efficiency, the Project Manager includes a dedicated function that returns users to the main menu at any given time; eliminating any need to restart the entire algorithm.

At the core of the Project Manager's Design (all other software cores included) is the use of persistent so-called while True loops, which govern the navigation flow through its sub functions such as creating, viewing and deleting projects. The loops ensure that upon completing any sub function, the user is seamlessly returned to the Project Manager's main menu. This workflow maintains a continuous user experience. Only when the user wants to return to the main menu, the algorithm

³⁹ Veenstra, J.P. (2025). AvantZero Sequence Diagram: Visual Representation on deleting an Avant Project and Returning to the Main Menu. [Image].

executes a break statement that exits the current loop, effectively stepping back one level within the navigational hierarchy. This hierarchical structure allows users to freely engage with any of the Project Manager's features (and functions found in all other cores) repeatedly and only exit to the higher-level menu when explicitly requested by selecting the 'return to main menu' option.

1. Users invoke the function call which starts the project deletion process. A list of existing projects is returned to the system core and subsequently visualized on the User Interface.
2. The user selects a project and confirms their choice. Their choice is sent directly to the *delete_project()* function. The function confirms one last time whether the user really wishes to delete the project. If confirmed, the project will be deleted. If cancelled, the user will be returned to the project manager's function.
3. If confirmed, the function will be completed by breaking the loop; returning the user to the project manager menu.
4. In the case that the user wants to return to the main menu, a simple break statement is sent to the sys core; breaking the loop found within the project manager. By breaking the loop, the user is sent back to the closest higher-level menu, which is the main menu of the algorithm.

2.3 – Data Generator

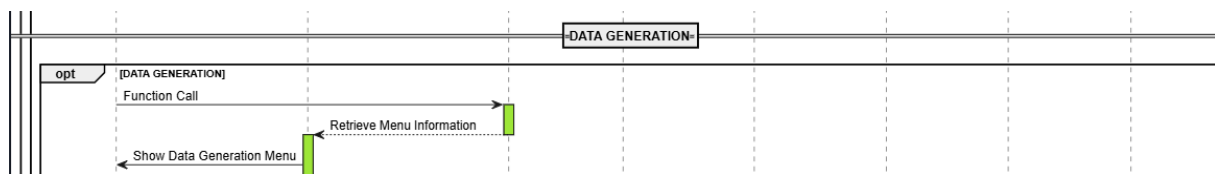


Figure 17: AvantZero Sequence Diagram: Visual Representation of the Data Generation Menu ⁴⁰

Just like the Project Manager, the Data Generator contains a sub menu containing various forms of functionality for the user to explore. The user can generate raw editing data (which is stored into JSON files for later reference) or use generated data to create EDL and XML files, generate depth of field maps from the footage timecodes embedded within the generated data, visualize or export all the data into dedicated PDF, CSV or image files.

Once the Data Generator is called, a function call is sent to the system and utilities cores to obtain the menu structure, contents and required helper functions. The information is returned to the system core, which visualizes it for the User Interface.

2.3.1 – Post-Production Data Generation

The process of generating post-production data and transforming the data into XML/EDL- based files for use within NLE software is deliberately broken down into two distinctive segments. This ensures that users can generate multiple sets of generated data for a singular bin of footage and create separate XML/EDL files for each of the generated data sets. It keeps Avant project concise and clear; especially if many avant projects have been created over the course of time. Users don't have to create new projects for each time they would like to generate data and make numerous copies of their footage bins.

⁴⁰ Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Data Generation Menu*. [Image].

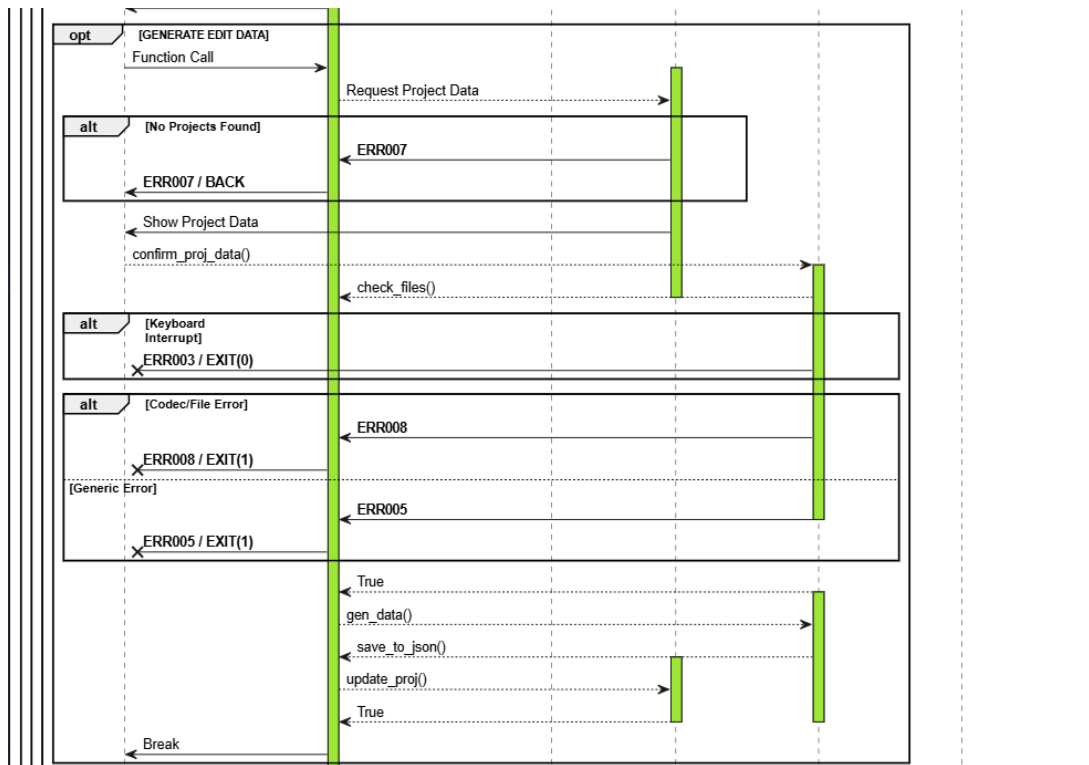


Figure 18: AvantZero Sequence Diagram: Visual Representation of the Edit Data Generation Menu ⁴¹

1. The user invokes a function call to start the Edit Data generator to the system core, and the project manager core is called to provide the User Interface with an overview of all existing Avant projects.
2. If no projects exist, an error code 007 is returned to the user. Afterwards, the user is returned to the Data Generation menu. If at any given time the user wishes to terminate the algorithm, they are free to do so with the CTRL + C shortcut, which immediately activates a Keyboard Interrupt within Python. Error code 003 is returned to the user alongside a visual notification that the algorithm will be terminated.
3. The user selects and confirms their chosen project. Once their choice is confirmed, the DATAGEN core is directly activated. The core conducts a check

⁴¹ Veenstra, J.P. (2025). AvantZero Sequence Diagram: Visual Representation of the Edit Data Generation Menu. [Image].

whether all files found within the project folder are compatible with the algorithm.

4. In the case of any kind of exception, the algorithm will return error code 005 and terminate immediately. If any of the files contain a codec that is currently unsupported by AvantZero, the algorithm will return error code 008 and immediately terminate the algorithm as well. This is done due to required manual action by the user: all unsupported files must be deleted from the project folder, and a manual restart of the algorithm is necessary before the function may be started again.
5. Whenever the check has been successfully completed, random edit data will be generated and saved to a json file for later reference. The algorithm utilizes the information specified by the user during the project setup (such as name, framerate, length and the like) and randomly generates a video sequence for the user. Considering new data has been generated, a hidden project file will be updated to reflect the current date and time. Whenever the user views their project with the project manager, they can see the exact date and times whenever data was generated.

2.3.2 – Generate EDL/XML

AvantZero requires three things to generate EDL and XML files: an Avant project folder, compatible media files and a generated dataset. The EDL/XML generator utilizes the generated dataset in JSON format to convert all the data from JSON to both XML and EDL formats for later use within various NLE suites, such as Adobe Premiere Pro, Blackmagic Da Vinci Resolve, Avid Media Composer or other NLE's that are compatible with XML and EDL datasets.

Once the EDL and XML's are generated, the user is able to view the visualized result. Of course, additional functionality exists, such as the ability to generate depth of field maps from the randomized timecodes of each of the footage to add further creative complexity to the compositions, or to export tangible copies of the generated data.

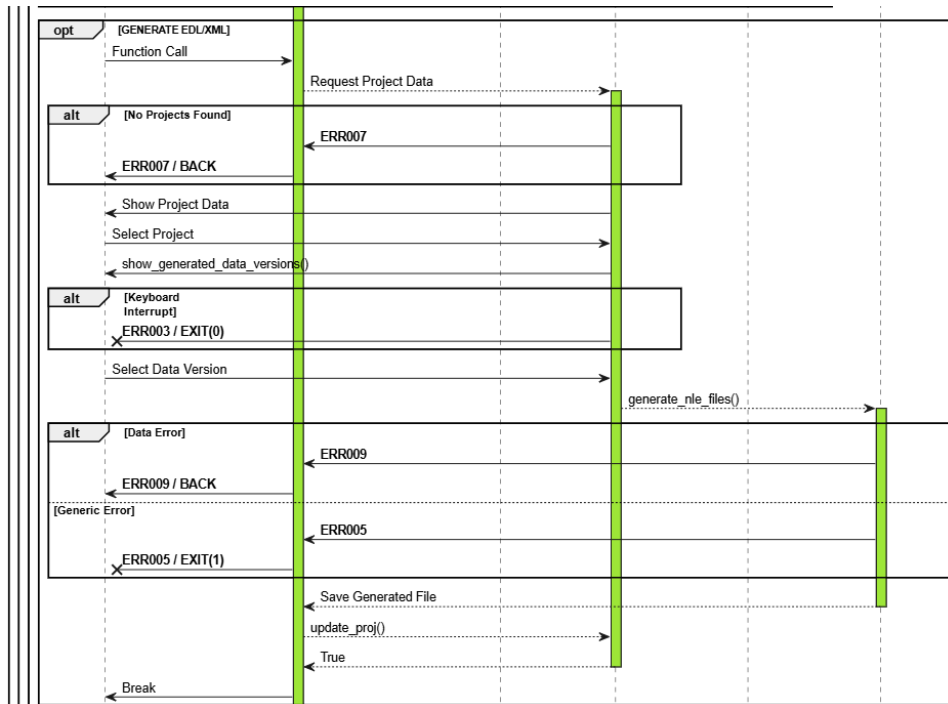


Figure 19: AvantZero Sequence Diagram: Visual Representation of the EDL/XML Generation Menu ⁴²

1. The EDL/XML generator starts the same as the data generator would. The user is requested to select their project of choice. Once the project has been selected, the user must select the version concerning the generated data they would like to export. Each version is labeled with a number, date and timestamp and a nickname created by the user.
2. Once everything seems to be in order, the NLE files are generated by the *datagen* core. In the case of a generic exception, or if there is any kind of problem with the JSON data (manual adjustments by the user or bugs that haven't been addressed yet), the algorithm will return either a error code 005 (and directly terminate) or error code 009 and return the user back to the data generator menu. Considering that small adjustments of the JSON data should, in theory (not taking any existing code-breaking bugs into account), be enough to fix the issues at hand and allow the user to simply retry the export process.

⁴² Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the EDL/XML Generation Menu*. [Image].

3. The project file is properly updated and now reflects the date and time when the project was exported to EDL/XML based files.

2.3.3 – Generate Depth

Generating depth of field maps is conducted separately from NLE data and EDL/XML files and considered secondary functionality. The depth generator will generate depth of field maps from the files and timecodes found within the JSON data with the DepthAnythingV2 library to mimic the NLE timeline from a depth of field perspective. Depth of field image streams can then manually be imported by the user to create new and exciting creative compositions.

At the time of writing, additional functionality, such as the ability to automatically generate a second XML based on the depth of field maps that are identical to the original footage, is currently considered for a minor patch but not actively set for further implementation. Therefore, importing depth of field maps to the composition will have to be conducted manually by the user, at least for now.

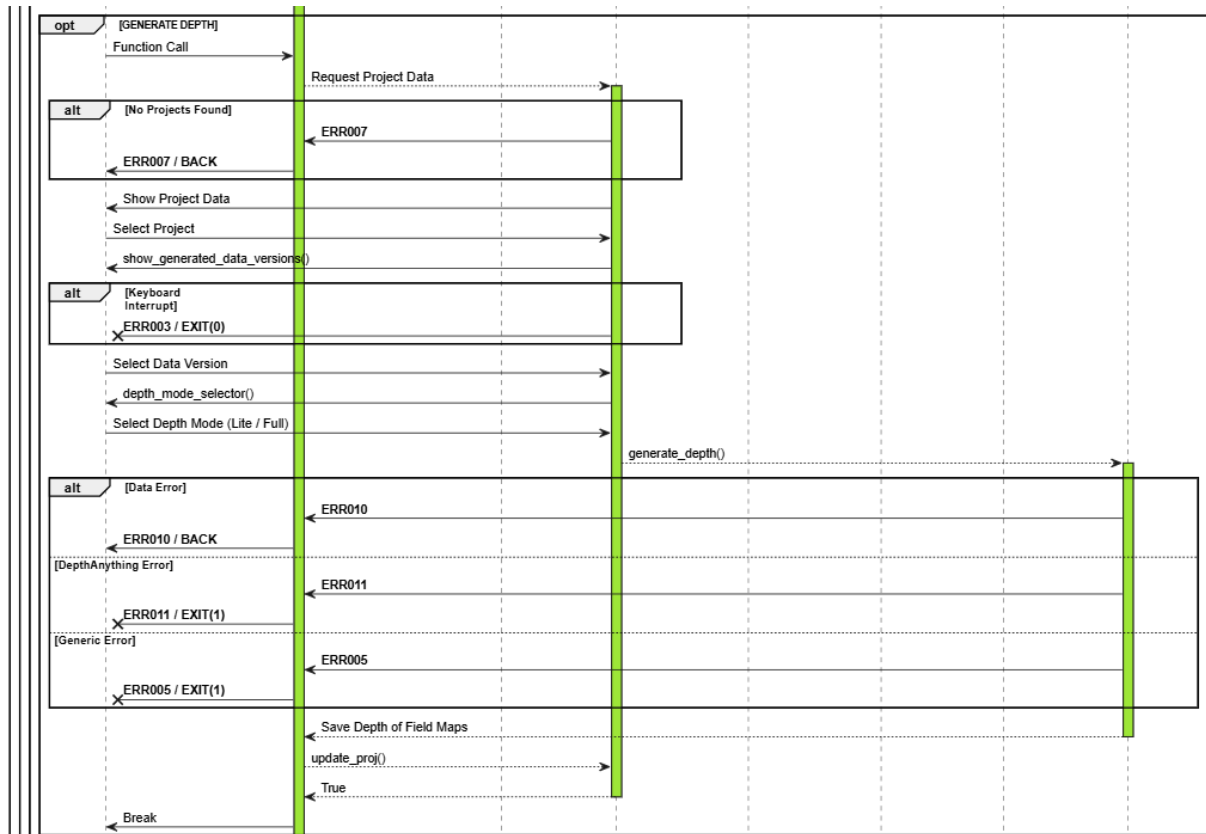


Figure 20: AvantZero Sequence Diagram: Visual Representation of the Depth Generation Menu ⁴³

1. The user selects their project and data version of choice. Once their choices have been confirmed, the *depth_mode_selector()* function will ask the user to select which mode the depth generator should use.
2. There are two distinct modes at play within the depth generator. A lite mode and full mode. The **lite** mode suffices for most of the depth of field map generation work: it will simply use the data found within the JSON file created by the data generator to create depth of field maps of each piece of footage and timecodes found to merely mimic the EDL/XML compositions in depth of field form. The **full** mode completely ignores the JSON data and generates depth of field maps of every single piece of footage found within the project folder at full length; providing the user with the ability to directly generate depth of field maps of their entire footage bin without having to generate post-

⁴³ Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Depth Generation Menu*. [Image].

production data. Please be advised that, in the case that the folder contains many files with longer runtimes, depth of field map generation could take a long time to complete depending on the specs of the computer the algorithm is run on.

3. The `generate_depth()` function is called and functionality found within the DEPGEN core and DepthAnythingV2 libraries are utilized to generate depth of field maps from the specified files. In case of a data error, the user will be returned to the data generation menu with error code 10. In all other cases, such as with general exceptions or errors found within the DepthAnythingV2 library, error code 005 (exceptions) or error code 011 (DepthAnything error) will be shown to the user, resulting in a direct termination of the algorithm.
4. Upon completion of the depth of field generation, all files are saved in the project folder, and the hidden project file is updated with date and timestamps of when the depth of field maps were generated. Afterwards, control is given back to the user and the algorithm automatically returns to the data generator menu.

2.3.4 & 2.3.5 – Data Visualization and Export

Each of the data sets generated can be visualized to several different graphs or better known as ‘plots’. Users can export different types of visualizations that each highlight different point of views of the data generated; the definitive list is still a work in progress at the time of writing. Considering AvantZero’s prototypical nature, the algorithm will only contain a handful of different plots, such as focus on how often a particular file was utilized and which timespan of each file was frequently used, to name a few examples. The full version of AvantZero, the Avant algorithm, will contain deeper layers of functionality and more forms of plotting.

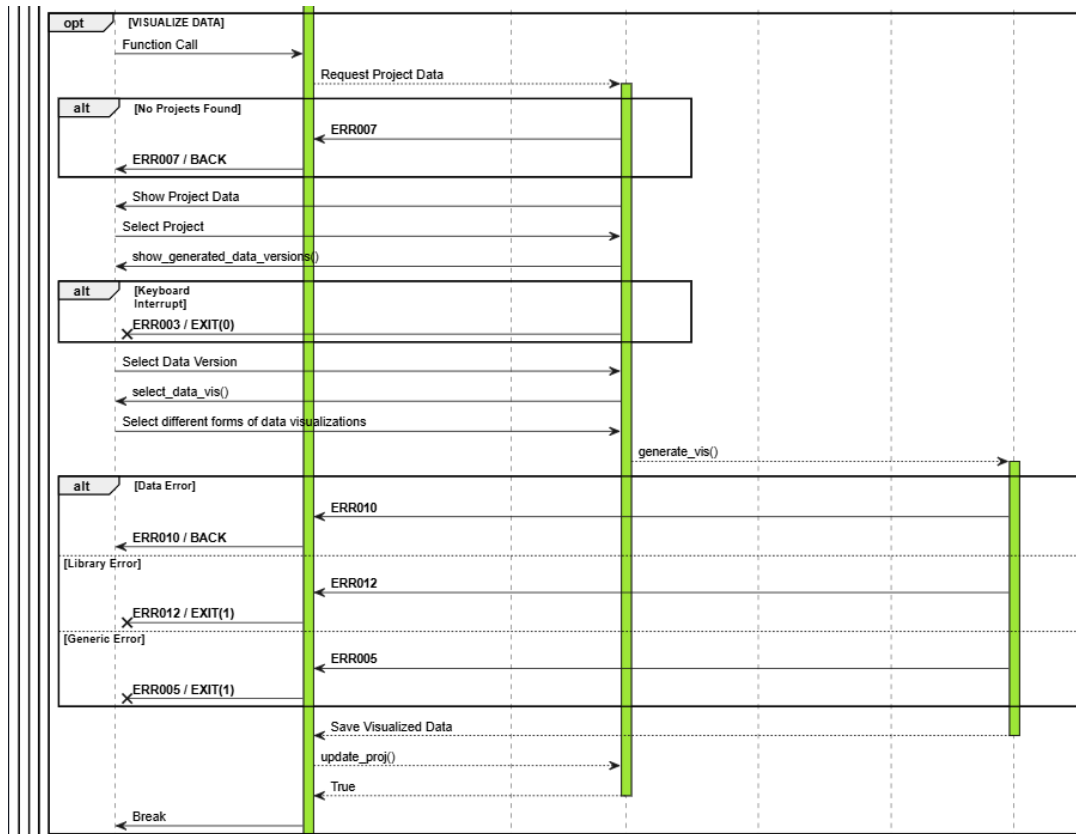


Figure 21: AvantZero Sequence Diagram: Visual Representation of the Data Visualization Menu ⁴⁴

1. After the user selects their preferred project and data version, the user is presented with a small menu that allows them to select one or multiple types of data visualizations.
2. Once confirmed, the `generate_vis()` function is invoked, subsequently activating the DANLY (data analysis) core. The matplotlib library is called and the requested visualizations are created. The hidden project file is updated with a date and timestamp to reflect that data has been visualized on this date.
3. In the case of a Data Error, error code 010 is shown to the user and the user is returned to the data generation menu. In any other case, such as a general exception or a library error, the algorithm will be unable to continue in any

⁴⁴ Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Data Visualization Menu*. [Image].

case, and the algorithm will be shut down as it requires manual intervention from the user.

AvantZero’s last form of data-related functionality lies in the exportation of generated data to tangible copies inside CSV files or PDF files. Both types of files have their advantages: PDF files provide a hard copy of all generated data for later reference, while the CSV files can be utilized for further analysis within Python, Jupyter Notebooks or other kinds of software and programming languages for users with that particular interest in mind.

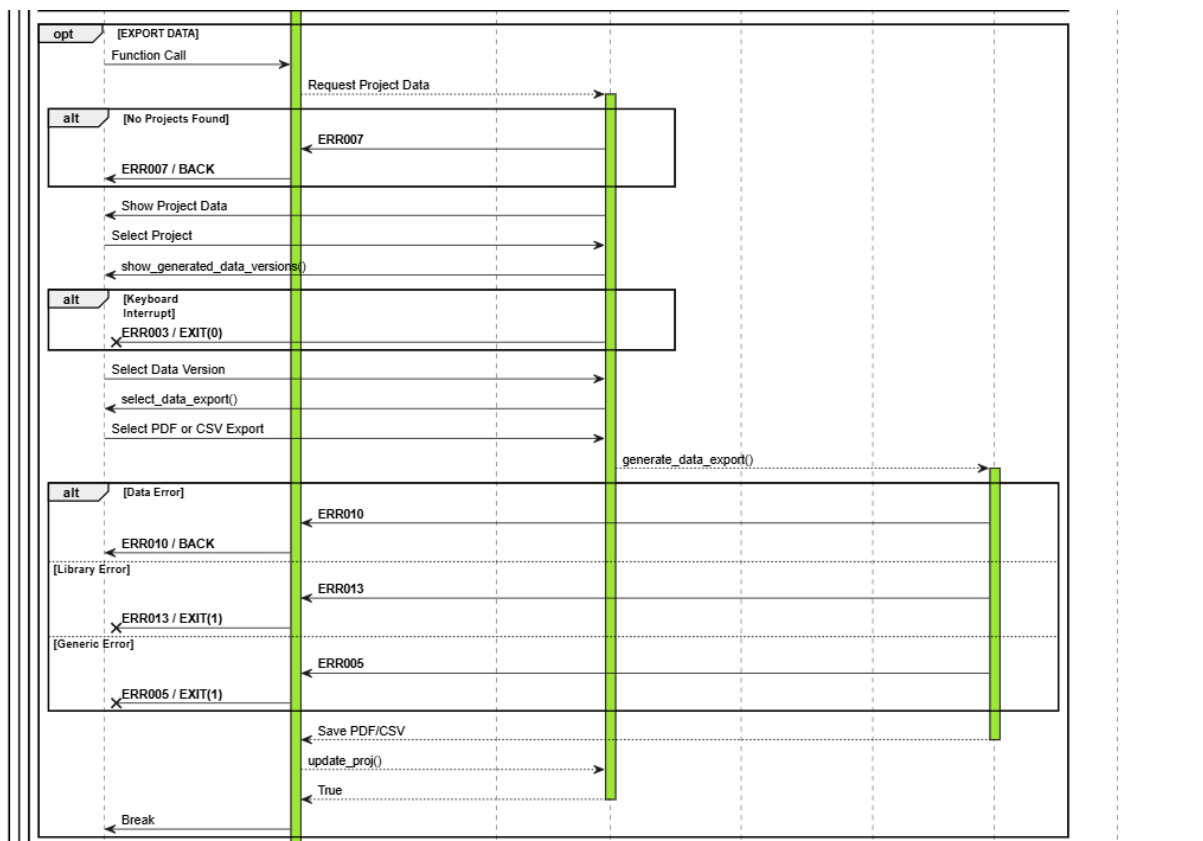


Figure 22: AvantZero Sequence Diagram: Visual Representation of the Data Export Menu ⁴⁵

⁴⁵ Veenstra, J.P. (2025). AvantZero Sequence Diagram: Visual Representation of the Data Export Menu. [Image].

2.4 & 2.5 – About AvantZero & Credits

Both the about and credits section contain similar forms of functionality and will be built to fairly the same detail. Each section contains hard-coded information regarding the algorithm such as background information, a direct link to the documentation website and extensive information of each of the libraries that have been implemented into the algorithm over the course of the development cycle. Therefore, both sections will be introduced singularly within the same chapter to avoid further redundancy.

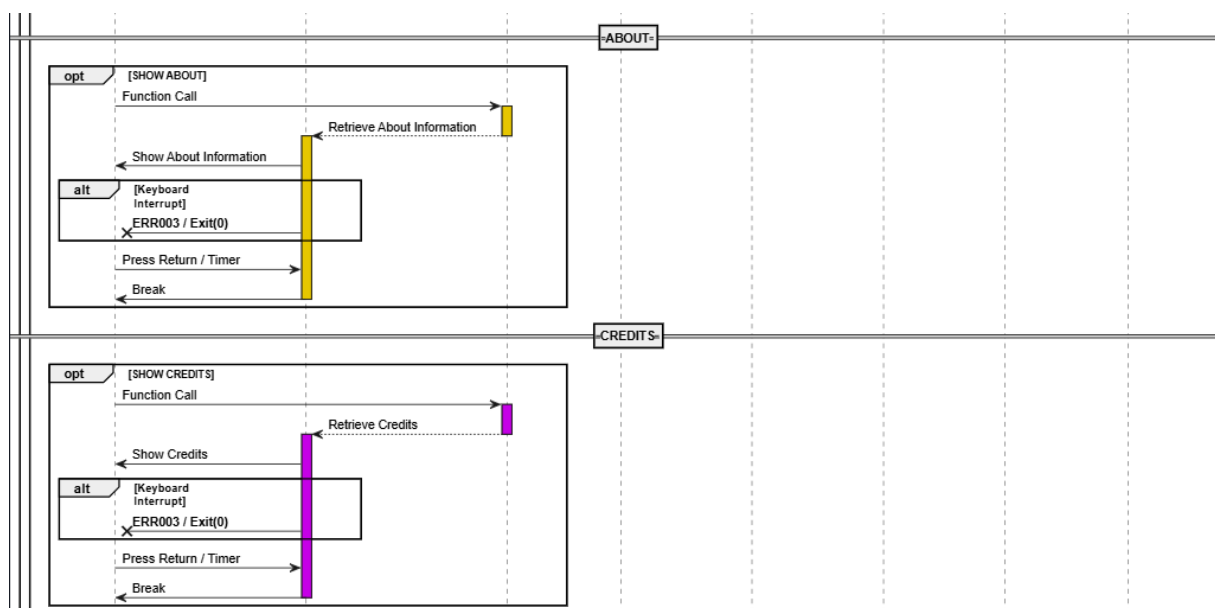


Figure 23: AvantZero Sequence Diagram: Visual Representation of the About and Credits Menu ⁴⁶

1. Both higher level functions in the main menu evoke their respective function calls and call upon the system and utilities cores. The utilities core contains all hard-coded text, which is retrieved by the system and visualized by the User Interface.
2. At any given time, the user is free to prematurely terminate the algorithm by pressing the CTRL + C shortcut (Keyboard Interrupt). A pop-up notification

⁴⁶ Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the About and Credits Menu*. [Image].

will inform the user that the Keyboard Interrupt has been activated. Afterwards, the algorithm will be directly and gracefully terminated.

3. Once all data has been presented on the screen (no matter whether it's the about or credits section), the user can return to the main menu of the algorithm by pressing the return key, thus breaking out of the loop.

2.6 – Exit

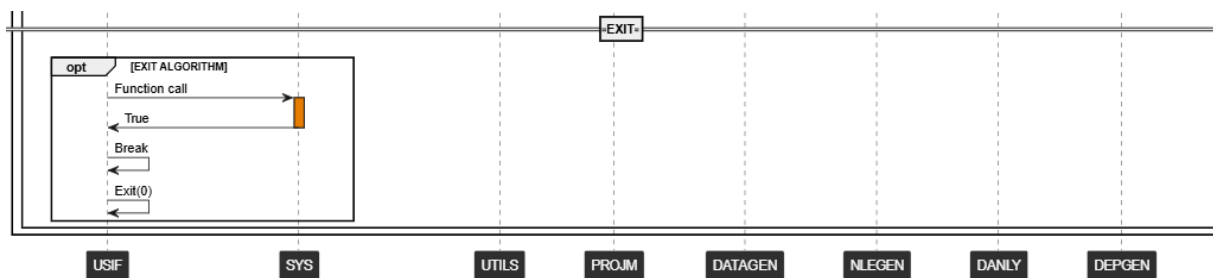


Figure 24: AvantZero Sequence Diagram: Visual Representation of the Exit function ⁴⁷

1. The final function found within the algorithm is the exit function. From here, the user can gracefully exit out of the algorithm and reallocate all the used memory and other resources back to the computer.
2. Once invoked, the exit function found in the system core will be called, returning a True value. The true value will simply cause the main menu loop (conducted through a while True loop) to break. Considering this was the last returnable loop within the algorithm, the algorithm will call upon the exit function with code 0. The algorithm is now terminated, and all resources are allocated back to the computer.

⁴⁷ Veenstra, J.P. (2025). AvantZero Sequence Diagram: Visual Representation of the Exit function. [Image].

Chapter 5 – Error Handling

In any interactive software system, especially one designed to accommodate user inputs and workflows, proper handling is both good practice and an absolute necessity. This is particularly true in the context of the AvantZero algorithm. It involves numerous scenarios in which user input is required for the system to proceed. These scenarios include but are not limited to project initialization, data selection, data visualization and depth of field generation.

User input is a double-edged sword in this case. While it offers extensive functionality and freedom to users, it may prove to be a highly unpredictable variable. No matter how well designed a user interface or detailed the instructions may be, there's always the possibility of users providing faulty input that the program cannot process properly; no matter if it is a formatting error, a misunderstanding of the requirements, or an entirely unexpected type of input. For instance, during project setup, a user might be prompted to input a faulty framerate; causing the program to crash or behave in unintended ways. The same risk applies during data visualization, where the user might reference to generated datasets that might have been altered or tampered with; causing improper formatting of the data and further misalignment with the algorithm.

To relieve such risks, the application should include scalable error-handling mechanisms that can detect and respond to unexpected behavior gracefully or terminate the algorithm in the safest way possible. This is a term I refer to as 'graceful termination'. Rather than allowing the program to fail silently or crash during every single mistake, we aim to implement structured error handles that serve both end users and developers alike.

Python provides a built-in way to handle errors using the *try-except* construct, with optional *else* and *finally* blocks to provide additional control over the execution flow of code. This pattern is used extensively throughout the algorithm to ensure that every interaction (especially those involving user input) is surrounded by fail safes.

Block	Definition
Try	Within the try block, we attempt to run code that may (or may not) potentially raise an error (or exception). Any function or input-handling logic that may be subject to failure is enclosed in this block.
Except	If an error were to occur inside the try block, the except block catches the error (or exception). This allows the program to respond to the error in a controlled way, such as logging it to the user.
Else	An optional block that runs only if no exceptions or errors were raised inside the try block. Else serves as a ‘follow-up’ block of try. If the try block can be executed without any errors, we could add any follow up code inside of the else block, such as saving results or retrieving information from a database.
Finally	The finally block will always be executed, regardless of whether the code ran successfully or raised an exception inside the try block. This block is often useful for releasing resources, disconnecting from databases, logging timestamps or performing cleanup actions.

Figure 25: A table containing the Python Try-Except structure ⁴⁸

⁴⁸ Veenstra, J.P. (2025). A table containing the Python Try-Except structure. [Table]. Unpublished Table.

This form of layered control does not only prevent unexpected crashes but provides future hooks for further debugging. By utilizing this structure, try-except patterns can be easily upscaled and expanded during later stages of the development cycle. The primary goal is clarity. If a user enters invalid data, they should receive understandable feedback that allows them to correct the mistake if possible. Emphasis is placed on ‘possible’ considering that during certain situations, terminating and restarting the algorithm is an inevitability as the situation requires either a proper restart or manual action from the user themselves.

To clarify diagnoses and documentation, error codes are used to tag specific categories of exceptions. These codes make it easier to track issues and serve as a point of reference for developers and end users in case something goes wrong. Several error codes have already been established and found within the sequence diagram described in the previous chapter. As development continues onward, additional errors codes and meanings may be added to the list. Down below outlines a table containing some of the errors that have been introduced within the sequence diagram, along with a brief description of each. A definitive list containing each of the possible errors that could occur within the AvantZero algorithm will be posted on the official documentation website.⁴⁹

Error Code	Meaning	Fix
001	<u>Dependency Error</u> : one or more cores are missing from the AvantZero folder	Redownload the missing files from the GitHub repository and place them in the AvantZero src folder.
002	<u>Package Error</u> : one or more required packages are either missing or cannot be initialized by Python	Retry to install all packages with pip install -r ‘requirements.txt’, or manually install each of the packages to see which one is missing; uninstall and reinstall the

⁴⁹ A Pixelated Point of View (2025) *AvantZero | Experimental Quake III Machinima Algorithm*. Retrieved August 11, 2025, from <https://avantzero-docs.vercel.app/>

		broken packages; if you are working inside a virtual environment: ensure that you have installed all packages inside the virtual environment; if you have multiple versions of Python installed: ensure you have selected the right Python interpreter.
003	<u>Keyboard Interrupt</u> : The user has manually terminated the algorithm with the CTRL + C shortcut	If the Keyboard Interrupt was unintended, the user will have to restart the algorithm.
004	<u>Permission Error</u> : AvantZero does not have gained permission from Windows to create folders and files within the my Documents folder	Ensure AvantZero is given permission to create folders and files by restarting the algorithm, running the algorithm as an administrator, if applicable, whitelist the algorithm in your antivirus software or check Windows permissions inside the Security tab (right click on my documents folder – ensure there's full control).
005	<u>Exception</u> : The algorithm has encountered a generic error (or exception) and must be closed	Restart the algorithm. If the error persists, open a issue on the GitHub repository.

Figure 26: A table containing various AvantZero Error Codes ⁵⁰

⁵⁰ Veenstra, J.P. (2025). A table containing various AvantZero Error Codes. [Table]. Unpublished Table.

Chapter 6 – Unit Testing

Proper testing of software ensures that new features work as intended, existing functionality remains stable, and potential issues are caught before they might impact the user. As AvantZero grows in scope and complexity, the importance of maintaining stability and clarity within the software cannot be overstated. Therefore, it is of vital importance that the project includes reliable test frameworks to conduct tests.

Each version of the algorithm will include a dedicated testing folder; found within the tests folder inside of the AvantZero repository. These tests are written to be automatable, while still leaving room for manual verification in cases where automation would be less efficient or simply overkill.

Two Python testing frameworks are in use for this project: **unittest** and **PyTest**. The unittest library is a standard testing framework found within Python, while PyTest is a popular third-party testing framework found on PyPi. Both frameworks enjoy extensive documentation and have their own strengths and use-cases. For example, unittest works best with classes and more verbose tests, such as when the stability of the algorithm is tested or when the software cores are generally tested. PyTest contains a more flexible fixture system and larger ecosystem and allows developers for quicker and deeper tests; making PyTest a perfect choice for more advanced and deeper functional checks.

Each of the many tests found within the repository will be repeatable, allowing those with interest in software testing and quality assurance to manually conduct each of the tests at any given time if desired. As the algorithm contains a broad roster of components, such as user interaction, data processing, visual output and file handling, various forms of testing will be conducted throughout the development cycle.

Unit Tests, for example, allow for individual functions or modules to be tested in isolation: ensuring further verification that input is correctly validated and that invalid data is properly rejected. With *Integration Tests*, multiple modules could be tested in isolation to confirm they are in fact able to work together. For example, integration tests could be utilized to check whether the Avant projects are setup properly and if the necessary folder structure and required files are rightfully created.

Error Handling Tests simulate invalid inputs or missing files to confirm that each of the errors is correctly generated and that each exception is indeed raised during the situation they were intended for; therefore potentially preventing any deeper issues regarding error handling to be nested further into the code. Lastly, *Manual Verification Tests* are conducted. These are manual tests that are conducted whenever they prove easier and faster for certain aspects of the software to be tested by hand. The main menu structure is a great example for Manual Verification Testing. It would turn out to save a lot of time in development whether the main menu is tested by hand (and whether each of the main functions boot properly), rather than to propose an entire test case in code, as we would only be interested in a simplified verification regarding the proper connection between two or multiple functions.

Testing results will be manually logged in each of the Python testing files, alongside additional information for users on how to recreate the tests for themselves. Each tests are stored within their respective version folder found within the testing environment on the GitHub repository: keeping the test environment clean, clear and traceable.

Chapter 7 – Development Sprints

Each aspect of AvantZero’s development cycle is broken down in smaller chunks to keep development clear and concise. In software development, these *chunks* are referred to as ‘development sprints’. A sprint is a defined period of time during which a set of tasks, features, or fixes are planned, executed, and reviewed. This structured approach ensures that even complex projects can advance in a predictable, organized manner.

A sprint consists of multiple tasks that are tied to a specific timeframe, such as a few days, five weeks, three months; even years for long-term goals. The tasks found within a specific sprint are based on various factors, such as: current importance, dependencies on other functionality, their position on the development roadmap or current availability of resources.

Each sprint is designed to achieve a clear set of objectives, such as the implementation of new features for the algorithm, the conduction of testing to ensure reliability, updated documentation or other technical references or the enhancement of existing functionality for performance, stability and/or user experience. Each of these goals are deliberately scoped so they can be realistically achieved within the sprint timeframe, while still making meaningful progress towards larger milestones found on the development roadmap.

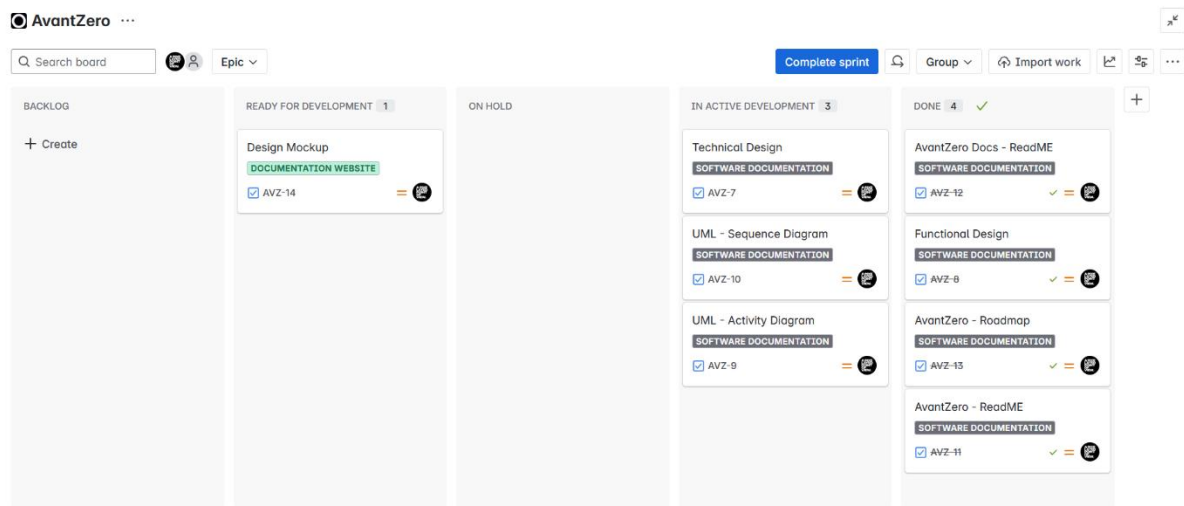


Figure 28: Image of the AvantZero Scrumboard: reflecting sprint 0 as of July 28th, 2025. ⁵¹

Each sprint is visualized through a *scrum board*: a digital board consisting of columns representing the different stages of work. The *backlog* contains all planned tasks for this sprint that have not yet been assigned for development. The *Ready for Development* section contains tasks that are ready to be started right away but have not been assigned. The *On Hold* section contains tasks that are currently placed on hold but, depending on the situation, could rapidly be resumed if necessary. All tasks found under the *In Active Development* column are currently assigned and worked on. Once these tasks have been successfully completed, they will be archived under the *Done* column to indicate that this task has been achieved in full. The sprint has been fulfilled once all tasks have been finalized within the set timeframe.

The AvantZero project contains an extensive overview of each development sprint on the GitHub repository. Under the *docs* section, each sprint will have their own folder, consisting of day-to-day updates with screenshots reflecting the current situation on the scrum board of each respective sprint. This visual approach allows for quick assessment of the status of the sprint at a single glance.

AvantZero currently consists of twenty-one sprints, excluding its initial release. Each sprint has its own subject or theme, during which a particular piece of functionality will be developed and tested. Sprint zero, for example, primarily focuses on the scope

⁵¹ Veenstra, J.P. (2025). *Image of the AvantZero Scrumboard, reflecting sprint 0 as of July 28th, 2025.* [Image].

and documentation of the algorithm, while sprint two solely focuses on the full completion of the algorithm's main menu. The full development roadmap can be found in the documentation section of the GitHub repository but will be briefly outlined below to highlight AvantZero's development cycle.

Sprint	Functionality
Sprint 0	Documentation, such as Functional Design, Technical Design, Readme, Sequence Diagram and the Development Roadmap.
Sprint 1	Completion of the file and folder structures. Furthermore, an outline of the main algorithm functions, cores and helper functions are defined. The documentation website has a fully working loading screen and homepage.
Sprint 2	The splash screen and main menu are fully working, tested and visualized. The user can reach each of the (for now) empty cores and gracefully terminate out of the algorithm.
Sprint 3	The documentation website has been updated with new information regarding the installation process of the algorithm, requirements and its background. The website is fully accessible, responsive and retrievable in two languages.
Sprint 4	Users can create, edit, select, view and delete Avant projects

Sprint 5	The algorithm contains basic functionality concerning data generation and can output EDL and XML files. The functionality is fully tested and implemented within the GUI.
Sprint 6	The algorithm can generate basic data with both video files and image sequences. All functions have been properly tested and implemented within the GUI.
Sprint 7	Users can export their data into PDF files. All functionality has been tested and implemented within the GUI.
Sprint 8	Users can export their data into both PDF and CSV files. All functionality has been tested and implemented within the GUI.
Sprint 9	Data can be visualized into plots and plots are exported to image files. Functionality is fully working and has been rigorously tested.
Sprint 10	The algorithm enjoys basic Depth of Field support for generating depth of field maps with the DepthAnythingV2 framework.
Sprint 11	The algorithm contains all of its Depth of Field related functionality.
Sprint 12	The Error Database on the documentation website is properly updated, is responsive, accessible and retrievable in two languages

Sprint 13	The license and credits pages on the documentation website have been completed
Sprint 14	The quick start guide on the documentation website has been fully completed
Sprint 15	The project manager page on the documentation website is finalized
Sprint 16	The Data Generator page on the documentation website has been fully finalized
Sprint 17	Work on the Data Visualization pages found on the documentation website have been completed
Sprint 18	Full documentation regarding depth of field functionality has been achieved.
Sprint 19	Alpha release of the algorithm. The code will be tested, and final adjustments are made
Sprint 20	Beta release of the algorithm. The code will be externally tested and made available on TestPyPi. Final adjustments are made for its release
Sprint 21	Initial release of the AvantZero algorithm.

Figure 29: Table with Avant Zero Development Sprints. ⁵²

⁵² Veenstra, J.P. (2025). *Table with AvantZero Development Sprints*. [Table]. Unpublished table.

Chapter 8: Deployment & Future Development

Considering AvantZero's prototypical nature, the algorithm will only receive minor patches and updates after its initial version 1.0.0 release. This means that new and extended forms of functionality will not be introduced to AvantZero beyond those that have been described within the development roadmap. Minor patches and updates are pushed to GitHub to ensure that AvantZero, in terms of functionality, will remain stable and ready for use in the upcoming future. Updates to the algorithm may contain, but are not limited to bug fixes, addition of video and image codecs that may be supported by the algorithm, visual changes and code optimizations.

Once the algorithm's development cycle is deemed complete (if there are no more outstanding issues nor direct fixes to implement), the repository will remain available as a publicly archived repository on GitHub and other internet services such as the Internet Archive to ensure the algorithm remains accessible for years to come.

In terms of deployment, the GitHub repository will host both the source code and executable-based releases. The documentation website is hosted on the Vercel network and a direct link to the documentation is provided both on the GitHub repository and as a shortcut inside of the AvantZero folder. For package distribution, AvantZero will be available on both PyPi and TestPyPi; the official Python Package Index and its testing counterpart. TestPyPi hosts all early, pre-release versions of AvantZero for testing purposes (which are not publicly visible). Official releases (version 1.0.0 and onward) will be available on PyPi and can be installed using a single console command.

Once installed through the PyPi package index, the algorithm can be initiated with a single command inside of a Command Line Interface window from anywhere on their computer. Any other case involves the user double-clicking the downloaded executable or using the Command Line Interface to navigate to the algorithm's source folder and starting the algorithm with the *python avantzero.py* command.

Chapter 9 – Limitations

The AvantZero algorithm knows a few limitations. This is considered normal concerning its background as prototypical software. For starters, the AvantZero algorithm will not receive several extended forms of functionality that Avant will have once its development cycle is considered complete: such as the implementation of machine-learning, fully automated and extended (post)production workflows, extended image and video codec support, external API connectivity (with the TRIII API – read as ‘Tree’) and a more sophisticated menu structure.

Secondly, the AvantZero algorithm can only create randomized compositions from *existing* footage to present a concept of the ‘RNG-based’ workflow it embraces, which serves as the main heart of both the Avant and AvantZero algorithms. AvantZero cannot capture randomized footage, which is functionality that Avant will have with the implementation of the Quake III Movie Maker’s Edition Mod.

Furthermore, AvantZero should be seen as a ‘GUI-first’ application as its development revolves around a Tkinter-based Graphical User Interface. The full version of AvantZero – the Avant algorithm – should be seen as a GUI/CLI hybrid as it offers two separate and dedicated modes for the user to choose between each mode upon boot: allowing the user to swap modes on the fly depending on their comfort level and familiarity with command line interfaces.

Lastly, the AvantZero will most likely not receive any major updates after its initial 1.0.0 release, except for a small number of patches and minor updates to ensure full functionality. Once AvantZero’s development lifecycle is deemed complete, the repository will be publicly archived on GitHub and other platforms such as the Internet Archive. Avant, on the other hand, will receive long-term support for updates, patches and further introduction of new functionality as its development cycle continues onward.

Chapter 10 - Looking Beyond: The Avant Algorithm

Once development on the AvantZero has officially ceased and the prototype is considered '*end of life*', development on the full version of AvantZero: the Avant algorithm, will be resumed in full and is expected to last a few more years before its initial release can commence. As the source code of both algorithms will most likely differ quite a bit, the first step in the development cycle is to ensure that Avant's code is up to the same level of quality and standards as that of its predecessor. In a sense, the code written for AvantZero serves as a blueprint for the full version of the algorithm.

Secondly, resources will be allocated to ensure that both the development roadmap, scrum boards, sprints, readme files and other forms of documentation for the algorithm (such as UML diagrams) are properly updated to reflect the definitive state of the algorithm. This includes the current state of the documentation website. While both AvantZero and Avant contain a different type of documentation website in terms of contents and design, both websites should utilize the same accessibility (a11y), internationalization (i18n) and responsivity standards to strive for maximum levels of user friendliness, clarity and accessibility on a multitude of different devices such as smartphones, laptops, tablets, foldables, monitors and television screens.

Once these factors have been properly updated, code-based development can further commence, and new functionality may be implemented and tested. One type of difference that set the two algorithms apart is that Avant can fully automate both the production and post-production of Quake III machinima while keeping the modular philosophy of AvantZero. Each function can be used consecutively after one another or as a stand-alone kind of fashion. Avant will have connectivity with an external API containing extensive information of hundreds of Quake III maps such as, but not limited to, scope, spawn point locations and camera space between the spawn point and the void. Avant can auto-download a randomized selection of Quake III maps or allow the user to select their own selection of maps.

The algorithm continues to generate Q3MME project files for the Q3MME mod and generate a bat file. The only thing the user will have to do is to execute the bat file to start the offscreen capturing process. Once the offscreen capturing has been

completed, the user can view their newly generated composition by importing the EDL or XML file into their NLE of choice; thus, fully providing users with an expansive and exciting new toolset to create algorithmic and highly experimental machinima of their own. Notions of machine learning will be implemented into the algorithm at a later stage, which will be extensively trained on the ever-expanding list of A Pixelated Point of View machinima. This way, users can choose a particular editing style found in machinima used within the training data set or have the algorithm choose one at random: adding a deeper layer of creativity and cinematography to each production.

In summary, the transition from AvantZero to Avant represents both a continuation and an evolution, building on proven foundations to deliver a more powerful, autonomous, and creatively algorithmic filmmaking tool. This next phase not only leverages the lessons learned and codebase established by AvantZero but also pushes the boundaries of automation, user customization, and machine learning integration. The journey ahead promises exciting possibilities for machinima creators and experimental filmmakers alike, offering them unprecedented control, efficiency, and creative freedom.

As Avant matures and enjoys deeper layers of complexity and functionality based upon the needs of machinima artists, it aims to become a highly valuable asset in the digital storytelling toolkit, empowering artists to explore new artistic frontiers and new, exciting algorithmic ways of production and experiences. The future holds vast and exciting potential and new experimental point of views for innovation, algorithmic collaboration, and artistic expression driven by the convergence of technology and creativity.

References

- Altar of Gaming (n.d.). *Quake III Arena*. Altar of Gaming. Retrieved July 16, 2025, from <https://altarofgaming.com/game/quake-iii-arena/>
- A Pixelated Point of View (2023). *Quake III Arena: Experimental Depth Map Use* [Video]. YouTube. Retrieved July 22, 2025, from <https://www.youtube.com/watch?v=7cu76qMIWIQ>
- A Pixelated Point of View (2025) *AvantZero | Experimental Quake III Machinima Algorithm*. Retrieved August 11, 2025, from <https://avantzero-docs.vercel.app/>
- Bittanti, M., & Veenstra, J.P. (in press). *Los Santos Plays Itself*. Mimesis International.
- Codecademy (n.d.). *What is CRUD?* Codecademy. Retrieved July 16, 2025, from: <https://www.codecademy.com/article/what-is-crud-explained>
- DepthAnything (2025). *Depth-Anything-V2*. [GitHub Repository]. GitHub. Retrieved July 22, 2025, from <https://github.com/DepthAnything/Depth-Anything-V2>
- DepthAnything (n.d.). *Depth Anything V2 project page*. GitHub Pages. Retrieved July 22, 2025, from <https://depth-anything-v2.github.io/>
- DepthAnything (n.d.) *Video Depth Anything: Consistent Depth Estimation for Super-Long Video's*. GitHub Pages. Retrieved July 22, 2025, from <https://videodepthanything.github.io/>
- entdark. (n.d.). *q3mme* [GitHub repository]. GitHub. Retrieved July 16, 2025, from: <https://github.com/entdark/q3mme>
- Goodwin, M. (2024). *What is an API (Application Programming Interface?)* IBM. Retrieved July 18, 2025, from: <https://www.ibm.com/think/topics/api>
- Hugging Face (n.d.). *Depth Anything V2 Demo*. Hugging Face. Retrieved July 22, 2025, from <https://huggingface.co/spaces/depth-anything/Depth-Anything-V2>
- Matplotlib Development Team (n.d.). *Matplotlib – Visualization with Python*. Matplotlib. Retrieved July 19, 2025, from <https://matplotlib.org/>
- Merriam-Webster. (n.d.). *Avant-garde*. In *Merriam-Webster.com dictionary*. Retrieved July 17, 2025, from <https://www.merriam-webster.com/dictionary/avant-garde>

Oxford Dictionary (n.d.). *Machinima*. In *Oxford English Dictionary*. Retrieved July 16, 2025, from https://www.oed.com/dictionary/machinima_n?tl=true

PlantUML (n.d.). *Quick Start Guide*. PlantUML. Retrieved July 28, 2025, from <https://plantuml.com/starting>

Python Software Foundation. (n.d.). *os – Miscellaneous operating system interfaces*. Python 3. Retrieved July 19, 2025, from <https://docs.python.org/3/library/os.html>

Python Software Foundation. (n.d.). *random – Generate pseudo-random numbers*. Python 3. Retrieved July 19, 2025, from <https://docs.python.org/3/library/random.html>

Python Software Foundation. (n.d.). *Requests: HTTP for Humans*. Python 3. Retrieved July 19, 2025, from <https://requests.readthedocs.io/en/latest/>

Python Software Foundation. (n.d.). *sys – System-specific parameters and functions*. Python 3. Retrieved July 19, 2025, from <https://docs.python.org/3/library/sys.html>

PyTorch Foundation (n.d.). *PyTorch*. PyTorch. Retrieved July 19, 2025, from <https://pytorch.org/>

Ramirez, S. (n.d.). *FastAPI*. FastAPI. Retrieved July 19, 2025, from <https://fastapi.tiangolo.com/>

Tensorflow (n.d.). *Tensorflow*. Tensorflow. Retrieved July 19, 2025, from <https://www.tensorflow.org/>

Veenstra, J.P. (2024). Dominion Algorithm [Bitbucket Repository]. *Bitbucket*. Retrieved July 17, 2025, from: <https://bitbucket.org/appov/dominion/src/main/>

Veenstra, J.P. (2025). *A table containing the Python Try-Except structure*. [Table]. Unpublished Table.

Veenstra, J.P. (2025). *A table containing various AvantZero Error Codes*. [Table]. Unpublished Table.

Veenstra, J.P. (2025). *Avant 0.2.9 running first-time setup* [Image].

Veenstra, J.P. (2025). *Avant 0.2.9 successfully completed first-time setup* [Image].

Veenstra, J.P. (2025). *AvantZero*. [GitHub Repository]. GitHub. Retrieved July 28, 2025, from <https://github.com/jiyorude/avantzero>

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of Initial Boot Functions* [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Dependency and Package Check Functions*. [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the boot_mame() function*. [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Project Manager Boot Process*. [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation on creating an Avant Project*. [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation on viewing an Avant Project*. [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation on modifying an Avant Project*. [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation on deleting an Avant Project and Returning to the Main Menu*. [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Data Generation Menu*. [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Edit Data Generation Menu*. [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the EDL/XML Generation Menu*. [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Depth Generation Menu*. [Image].

Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Data Visualization Menu*. [Image].

- Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Data Export Menu*. [Image].
- Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the About and Credits Menu*. [Image].
- Veenstra, J.P. (2025). *AvantZero Sequence Diagram: Visual Representation of the Exit function*. [Image].
- Veenstra, J.P. (2025). *Dominion Algorithm: Code Snippet* [Image].
- Veenstra, J.P. (2025). *Dominion Algorithm Repository* [Image].
- Veenstra, J.P. (2025). *Image of the AvantZero Scrumboard, reflecting sprint 0 as of July 28th, 2025*. [Image].
- Veenstra, J.P. (2025). *Main menu of the Avant algorithm. Version 0.2.9* [Image].
- Veenstra, J.P. (2025). *Partial Representation of AvantZero Sequence Diagram* [Image].
- Veenstra, J.P. (2025). *Table with AvantZero Development Sprints*. [Table].
Unpublished table.
- Veenstra, J.P. (2025). *Using Depth Anything V2 on Hugging Face* [Image].
- Yang, L., Kang, B., Huang, Z., Zhao, Z., Xu, X., Feng, J., & Zhao, H. (2024). Depth Anything V2. In *Proceedings of NeurIPS 2024*. arXiv. Retrieved July 22, 2025, from <https://arxiv.org/abs/2406.09414>