Pycraft's Documentation

Release 9.5.7

Thomas Jebbo

Jan 30, 2023

CONTENTS

Pycraft is an OpenGL, open world, video game made with Python.

CHAPTER

ONE

INTRODUCTION

1.1 About

Pycraft is a 3D open-source, open-world video game made in Python. For a long time attempts to make large 3D games in Python have been ignored, we believe there are two reasons: one; People use Python primarily for data handling and processing and not graphics and, two; there is little to no documentation out there to do anything more than make a 3D rotating cube in Python. Making a 3D game in Python for us hasn't been an easy experience, far from it but we have decided to share my project, complete with tutorials, explanations, articles and code explanations in the hope that 3D game development in Python can be seen as a more easily attainable target, and to fill that gap in documentation. Pycraft then is a trial project, as we learn and experiment on what goes best where and how thing go together, this is why development can sometimes appear to have stopped, because we are learning and testing what we have learned, so hopefully for people in the future it will be an easier experience. Also, don't forget there is more to game development than just graphics, there is AI, sound, physics and all the other GUIs that go with it, and as we learn the quality of the overall program will improve. Pycraft is not going to be the final name of the game, however until something better becomes available, we shall stick to it.

1.2 Setup

1.2.1 Installing the project from GitHub (Method 1)

The project will download as a (.zip) compressed file. Please make sure you have the project decompressed before use. Next make sure that any folders and files outside of the 'Pycraft' folder are removed and that the 'Pycraft' file is in the intended place for the file to be run from. This file can be freely moved around, transported between drives, computers and folders in this form!

*Just make sure that if you plan to use the installer that you make sure the file location is correct after you have moved the project, to do this simply remove everything in the 'pycraft/Data*Files/InstallerConfig.json' file and re-load the game, it will try to repair the file and write the new path instead, during this process it may appear that Pycraft has crashed as it will likely bring up an error message, a more user-friendly experience is coming soon**

When running the program please make sure you have a minimum of 1GB of free space on the drive and also have Python 3.7 or above installed on your device. This can be found here: (www.python.org/downloads). The version of Python isn't too important in this circumstance however the project has been tested in Python 3.7 and above and is known to work. In addition to all this please make sure you have the following modules installed on your device:

Pygame, Numpy, Pillow, PyAutoGUI, Psutil, PyWaveFront, CPUinfo, Ctypes, ModernGL, ModernGL*Window, GPUtil, Pyrr, PyJoystick, Noise and Matplotlib.

For those not familiar they can be found here: (pypi.org).

You can use the following syntax to install, update and remove these modules:

`pip install <module>` `pip uninstall <module>`

Here is a short video tutorial walk you through all this: (https://youtu.be/DG5YbE-umw0)

1.2.2 Installing the project from GitHub (Method 2)

If you are installing the project from the GitHub releases page or through Source Forge, then this will be relevant for you. After you have selected your preferred file type (it'll be either a compiled (.exe) file or a (.zip) file, those that download the (.zip) file will find the information above more relevant.

If you, however, download the (.exe) type file, then this will be more relevant for you. If you locate the file in your file explorer and double click it, then this will run the project. You do not need Python, or any of the projects required modules, as they come built-in with this method. This method does also not install anything extra to your devise, to remove the project, simply delete the (.exe) file in your file explorer. Please note that it can take a few moments for everything in the (.exe) file to load and initialise, so nothing might not appear to happen at first. Also, you can only run one instance of Pycraft at any time (even if you are using another method).

1.2.3 Installing from PyPi (preferred)

If you are installing the project from PyPi, then you will need an up-to-date build of Python (3.7 or greater ideally) and also permission to install additional files to your device. Then you need to open a command-line interface (or CLI), we recommend Terminal on Apple based devises, and Command Prompt on Windows based machines. You install the latest version of Pycraft, and all its needed files though this command:

```
pip install Python-Pycraft
```

and you can also uninstall the project using the command:

```
pip uninstall Python-Pycraft
```

And now you can run the project as normal. Please note that at present it can be a bit tricky to locate the files that have downloaded, you can import the project into another python file using:

```
import Pycraft
```

1.2.4 Installing using Pipenv

You can alternatively run these commands in the directory containing a file called *Pipfile*:

`pip install pipenv` then: `pipenv install python-pycraft`

And to start the game: `pipenv run python <PATH to 'main.py'>`

1.3 Running The Program

When running the program, you will either have a (.exe) file, downloaded from the releases page, or you will have the developer preview, if you have the developer preview, which can be found in the files section of this repository then this is how you run that program.

Now you have the program properly installed hopefully (you'll find out if you haven't promptly!) you need to locate and run the file "main.py" if it crashes on your first run then chances are you haven't installed the program correctly, if it still doesn't work then you can contact us. We do hope however that it works alright for you and you have a pleasant experience. This program has been developed on a Windows 64-bit computer however should run fine on a 32-bit Windows machine (uncompiled) or through MacOS although they remain untested for now.

We recommend creating a shortcut for the "main.py" file too so it's easier to locate.

1.4 Credits

1.4.1 With thanks to;

- Tom Jebbo (PycraftDeveloper) @ www.github.com/PycraftDeveloper
- Count of Freshness Traversal @ www.twitter.com/DmitryChunikhin
- Dogukan Demir (demirdogukan) @ www.github.com/demirdogukan
- Henri Post (HenryFBP) @ www.github.com/HenryFBP
- PyPi @ www.pypi.org
- PIL (Pillow or Python Imaging Library) @ www.github.com/python-pillow/Pillow
- Pygame @ www.github.com/pygame/pygame
- Numpy @ www.github.com/numpy/numpy
- PyAutoGUI @ www.github.com/asweigart/pyautogui
- Psutil @ www.github.com/giampaolo/psutil
- PyWaveFront @ www.github.com/pywavefront/PyWavefront
- Py-CPUinfo @ www.github.com/pytorch/cpuinfo
- GPUtil @ www.github.com/anderskm/gputil
- Tabulate @ www.github.com/p-ranav/tabulate
- Moderngl @ www.github.com/moderngl/moderngl
- Moderngl*window @ www.github.com/moderngl/moderngl-window
- PyJoystick @ www.github.com/justengel/pyjoystick
- Matplotlib @ www.github.com/matplotlib/matplotlib
- FreeSound: - Erokia's "ambient wave compilation" @ www.freesound.org/s/473545
- FreeSound: - Soundholder's "ambient meadow near forest" @ www.freesound.org/s/425368
- FreeSound: - monte32's "Footsteps*6*Dirt*shoe" @ www.freesound.org/people/monte32/sounds/353799
- Freesound: - Straget's 'Thunder' @ www.freesound.org/people/straget/sounds/527664/
- Freesound: - FlatHill's 'Rain and Thunder 4' @ www.freesound.org/people/FlatHill/sounds/237729/
- Freesound: - BlueDelta's 'Heavy Thunder Strike - no Rain - QUADRO' @ - www.freesound.org/people/BlueDelta/sounds/446753/
- Freesound: - Justkiddink's 'Thunder » Dry thunder1' @ www.freesound.org/people/juskiddink/sounds/101933/
- Freesound: - Netaj's 'Thunder' @ www.freesound.org/people/netaj/sounds/193170/
- Freesound: - Nimlos' 'Thunders » Rain Thunder' @ www.freesound.org/people/Nimlos/sounds/359151/
- Freesound: - Kangaroovindaloo's 'Thunder Clap' @ www.freesound.org/people/kangaroovindaloo/sounds/585077/
- Freesound: - Laribum's 'Thunder » thunder*01' @ www.freesound.org/people/laribum/sounds/353025/
- Freesound: - Jmbphilmes's 'Rain » Rain light 2 (rural)' @ www.freesound.org/people/jmbphilmes/sounds/200273/

1.5 Uncompiled Pycraft Dependencies

When you're installing the uncompiled Pycraft variant from here you need to install the following 'modules', which can be done through your Control Panel in Windows (First; press <windows key + r> then type "cmd" then run the below syntax) or on Apple systems in Terminal.

`pip install <module> pip uninstall <module>`

pip is usually installed by default when installing Python with most versions.

- PIL (Pillow or Python Imaging Library) @ www.github.com/python-pillow/Pillow
- Pygame @ www.github.com/pygame/pygame
- Numpy @ www.github.com/numpy/numpy
- PyAutoGUI @ www.github.com/asweigart/pyautogui
- Psutil @ www.github.com/giampaolo/psutil
- PyWaveFront @ www.github.com/pywavefront/PyWavefront
- Py-CPUinfo @ www.github.com/pytorch/cpuinfo
- GPUtil @ www.github.com/anderskm/gputil
- Tabulate @ www.github.com/p-ranav/tabulate
- Moderngl @ www.github.com/moderngl/moderngl
- Moderngl*window @ www.github.com/moderngl/moderngl-window
- PyJoystick @ www.github.com/justengel/pyjoystick
- Matplotlib @ www.github.com/matplotlib/matplotlib

Disclaimer; unfortunately, lots of these python modules (first and third party) can require some external modules that will be installed during the installing process of the above modules, unfortunately this makes it really difficult to give credit to those modules, if you have any recommendations, please contact me appropriately.

1.6 Changes

Pycraft v9.5.5 is now live! Here is a list of all the added features to this minor update:

- Feature: We have extensively reworked the directory structure of Pycraft to make it more user friendly and easier to find and access necessary files.
- Feature: Pycraft has been entirely restructured to reduce the reliance on the 'self' parameter to make Pycraft's source code easier to work with.
- Feature: We have simplified events in Pycraft now so that they all use the same method of detecting them regardless of if you're using Pycraft's 2D or 3D engine.
- Feature: We have changed the 3D windowing engine to match the 2D windowing engine to bring feature parity and to make the transition between windowing engines easier. By doing this we managed to improve in game performance, significantly simplify the method of sharing data between windowing engines, allow changes to the new settings menu to control more of the 3D engine, and to allow changes to the settings in the settings menu to be applied to the 3D engine without necessitating a restart.
- Feature: We have added back in the Loading, Inventory and Map UIs, and all of them have been extensively reworked and changed to be more featureful and behave better with the new 3D engine.
- Feature: A new dropdown element in the settings menu has been added.

- Feature: We have used the new dropdown element for the settings menu to add in translations and adjustments to the rendering resolution of Pycraft.
- Bug-Fix: we have finished one of the most extensive pre-release testing processes yet - due to the large number of changes we have made - and fixed a variety of known bugs, with a particular focus on the 3D engine, controller compatibility and the installer.
- Documentation: We have started the process of adding in docstrings to the start of every class, function and procedure in Pycraft, and later this will extend to also include at the start of each module.
- Documentation: We have completely restarted the documentation for Pycraft and will be using a new automated method to make the process of compiling the new docstrings together and formatting them properly, in addition to formatting this ReadMe automated for future ease of use. This has yet to be publicly released though.

Again, feedback would be much appreciated this update was released on; 23/12/2022 (date format; DD/MM/YYYY). As always, we hope you enjoy this new release and feel free to leave feedback.

1.7 Understanding the release notes

This section will hopefully provide additional information on helping to read the release notes.

- Points detailed after the "Feature" tag are what was focused on in the update and will likely always be present in each update, often this is the most significant area of the update.
- Points detailed after the "Bug-Fix" tag are likely to be the most frequent, they outline the most major bugs that have been fixed in this update, although they are not the only bugs that have been fixed.
- Points detailed after the "Performance" tag are used where there have been significant performance improvements to the project.
- Points detailed after the "Identified-Bugs" tag are bugs that have been identified in the project and that haven't been fixed as of writing the release notes, these are significant issues and will be fixed as soon as possible.
- Points detailed after the final "Documentation" tag are indicators of significant improvements to the documentation. The "PEP8" tag is used to signify that significant changes have been made to Pycraft to bring it in line with the PEP8 standards.

1.8 Input mapping

This section will be replaced with a dedicated file for keymapping as well as an in-game guide when this area of Pycraft is completed.

1.8.1 Keyboard

- Use W, A, S, D in game to move around, and use these keys in the map GUI to move that around.
- Use SPACE to jump in game, reset your zoom in the map GUI, start the benchmark section, or press 10 times to enter Devmode.
- Use E in game to access your inventory
- Use R in game to access the map
- Use F11 to toggle full-screen
- Use Q to access a resource value screen

- Use L in game to toggle locking your mouse (forcing it to stay in the window or not)
- Use X to exit Devmode

1.8.2 Mouse

- SCROLL in the map to zoom in/out, or to scroll the settings menu
- LEFT CLICK to select

A detailed map of inputs for keyboard and mouse or controller combinations is coming; for now, see the section below, toggling between full-screen is currently not bound to a button on the controller because we will need all the different buttons for gameplay

1.9 Our Update Policy

New releases will be introduced regularly, it is likely that there will be some form of error or bug, therefore unless you intend to use this project for development and feedback purposes (Thank you all!) we recommend you use the latest stable release; below is how to identify the stable releases.

1.10 Version Naming

Pycraft's versions will always now follow the structure; "vA.B.C"

- Where "A" is the major revision number.
- Where "B" is the minor revision number.
- Where "C" is the patch and developer preview numbers (combined).

Every version of Pycraft as of the 27/10/2022 (DD/MM/YYYY) must feature all 3 values. Updates also now go sequentially, so Pycraft v9.6.4 is newer than Pycraft v9.5.7. If either of the "A" or "B" version numbers is incremented in a release, documentation MUST be suitably updated, in addition Pycraft MUST be released on PyPi, SourceForge and as a release on GitHub.

1.11 Releases

All past versions of Pycraft are available under the releases section of Pycraft, this is a new change, but just as before, major releases like Pycraft v0.9 and Pycraft v0.8 will have (.exe) releases, but smaller sub-releases will not, this is in light of a change coming to Pycraft, this should help with the confusion behind releases, and be more accommodating to the installer that's being worked on as a part of Pycraft v0.9.4. This brings me on to another point, all past updates to Pycraft will be located at the releases page (Thats all versions), and the previous section on the home-page with branches will change. The default branch will be the most recent release, then there will be branches for all the sub-releases to Pycraft there too; and the sister program; Pycraft-Insider-Preview will be deprecated and all data moved to relevant places in this repository, this should hopefully cut down on the confusion and make the project more user-friendly.

1.12 Other Sources

We now post a roughly monthly article about Pycraft, showing behind the scenes, tips and tricks and additional information, this is shared to both Medium (medium.com/@PycraftDev) and Dev (dev.to/PycraftDev) and builds on the regular posts we share to Twitter (twitter.com/PycraftDev) and Dev (dev.to/PycraftDev).

1.13 Final Notices

Thank you greatly for supporting this project simply by running it, we are sorry in advance for any spelling mistakes. The program will be updated frequently and we shall do my best to keep this up to date too. we also want to add that you are welcome to view and change the program and share it with your friends however please may we have some credit, just a name would do and if you find any bugs or errors, please feel free to comment in the comments section any feedback so we can improve my program, it will all be much appreciated and give as much detail as you wish to give out.

CHAPTER

TWO

FORMATTING GUIDE

2.1 Introduction

Welcome to Pycraft's new formatting guide! In this guide you will learn more about the structure that we follow in the source code for Pycraft. To start with, it is not mandatory that this is read before you start work on Pycraft, and nor os it a replacement for the excellent PEP-8 Guide for code formatting in Python - it is however an extension of this concept. This guide aims to maintain a constant style of formatting through Pycraft and its sister projects (for example the Installer), in order to make it easier to transfer from code written by one developer to another. If you are confused by the structure of Pycraft, or by something in this guide then don't hesitate to get in touch (ways of how to do so can be found in the Introduction tab). Each area in this guide will feature a description, an example (likely not taken from Pycraft for simplicity) - where relevant, our reasoning for this decision, any additional information regarding this formatting feature in Pycraft's source code and be preceeded by a text description about what that section of the guide is for.

2.2 Docstring Formatting Guide

Welcome to Pycraft's docstring formatting guide. Docstrings are a relatively recent addition to the source code of Pycraft - only being introduced in Pycraft v9.5.5 as structural changes allowed for the use of docstrings to be effective -, and act as text descriptions and overviews of the function of that area of code, as well as that area's required parameters, keyword arguments and outputs. Currently any subroutine (procedure or function) and class must have a docstring, with the exception of a class instantiation method with no function; which can be identified as being identical to:

```
def __init__(self):
    pass
```

It is crucial that for docstrings the correct format is used as the documentation gets built using an automated method and incorrect foratting can create problems. So here are the set of rules for correctly formatting a docstring in Pycraft:

- All docstrings for Pycraft must be written in English (US ideally, but UK is fine too).
- Any dates must be given with the format used to create them, for example 'released on the date: 12/07/2022 (using the DD/MM/YYYY format)'
- A docstring must - regardless of location - consist of a description that acts as an overview of what that code does. There is no limit to the length of the description, although ideally it should be as detailed as possible so that any reader does not need to check the actual code after reading the docstring description for futher information. Additionally try to avoid just rewriting the code in plain english.
- Having a docstring at the start of a module in Pycraft isn't currently required, however this is constantly being reviewed.

- The docstring corresponding to a class must not have a '- Output:' section. However the docstring for any subroutine (except for the default instantiation method) must contain the ' - Output:' description.

2.3 Module Formatting Guide

2.4 Class Formatting Guide

2.5 Subroutine Formatting Guide

2.6 Variable and Constant Formatting Guide

2.7 Shader Formatting Guide

2.8 Directory Formatting Guide

CHAPTER

THREE

MODULE BREAKDOWN

3.1 OpenGL_window_benchmark

3.1.1 run_opengl_window_benchmark

This class is in charge of the OpenGL window benchmark seen in the benchmark section of Pycraft.

- Args:
  - None
- Keyword Args:
  - None

setup

This subroutine is in charge of loading the resources required by the OpenGL benchmark, including: 1x Texture 1x 3D Scene 1x GLSL Shader And also sets the window parameters so that the OpenGL benchmark sets up in the same way on all devices for consistency.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
  - wnd (BaseWindow): This is used by ModernGL_window as the display object to use for rendering and additional resource loading.
- Keyword Args:
  - None
- Output:
  - texture (ModernGL_window Texture): This texture is rendered to the scene to add additional complexity.
  - mvp (ModernGL_window Shader Attribute): This matrix is used to render the position and rotation of the scene.
  - light (ModernGL_window Shader Attribute): This attribute is used to shade the scene based on the position of the camera.
  - vao (ModernGL VertexArray): This is the scene we render (a cube).
  - timer (float): This is used to keep track of how long this section of the benchmark has been running for, and is used in calculating the rotation of our scene.

  - aspect_ratio (float): This float represents the aspect ratio we want our display to be rendering at.

start

This subroutine is used to render the OpenGL window benchmark, accessible when run through the benchmark section of Pycraft. This test is the final of 3 tests designed to test different aspects of your hardware. This stresses your GPU as well as your CPU and this is often the most difficult benchmark to run.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
  - iteration (int): In the benchmarking process, iteration is used to keep track of how long the benchmark has been running
  - Setfpslength (int): This is the length of the 'Setfps' array, we use this instead of specifying an integer in order to allow us to make changes later on in Pycraft's development about how many targets to use for the benchmark section.
  - Setfps (array): This is an array of integers that stores FPS targets for the benchmark section of Pycraft, with each element being a different FPS to try to reach, getting progressively harder. The FPS from this array is updated every 500 iterations of the benchmark.
  - fpscounter (int): This is used to store the index used to calculate the next element in the 'Setfps' array, this is used so Pycraft know's what to set the FPS to next, and what to set the caption to so that it displays the current FPS being tested.
  - Maxiteration (int): This is used to calculate after how many iterations we move onto the next targeted FPS, currently this is set to increase the FPS every 500 'iteration's.
  - ctx (Context object): This is used by ModernGL for loading OpenGL resources and enabling access to OpenGL features.
  - texture (ModernGL_window Texture): This texture is rendered to the scene to add additional complexity.
  - mvp (ModernGL_window Shader Attribute): This matrix is used to render the position and rotation of the scene.
  - light (ModernGL_window Shader Attribute): This attribute is used to shade the scene based on the position of the camera.
  - vao (ModernGL VertexArray): This is the scene we render (a cube).
  - timer (float): This is used to keep track of how long this section of the benchmark has been running for, and is used in calculating the rotation of our scene.
  - aspect_ratio (float): This float represents the aspect ratio we want our display to be rendering at.
- Keyword Args:
  - None
- Output:
  - fpslistX (array): Used to store the iteration of the benchmark. This correlates to a point, with this making up the X coordinate and 'fpslistY' making up the Y coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.
  - fpslistY (array): Used to store the FPS at a given iteration of the benchmark. This correlates to a point, with this making up the Y coordinate and 'fpslistX' making up the X coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.

3.2 __init__

3.3 achievements

3.3.1 generate_achievements

This class is in charge of rendering of the achievements GUI.

- Args:
  - None
- Keyword Args:
  - None

achievements_gui

This subroutine does the bulk rendering of the achievements GUI.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

3.4 benchmark

3.4.1 generate_benchmark

This class does the bulk of the rendering for the benchmark start and results section and also manages the running and execution of the benchmark.

- Args:
  - None
- Keyword Args:
  - None

benchmark_gui

This subroutine does the bulk of the rendering for the benchmark start and results section and also manages the running and execution of the benchmark.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

3.5 benchmark_utils

3.5.1 close_benchmark

This class is in charge of switching back from the benchmark UI to Pycraft and making sure that the benchmark engine is reset so it behaves as expected next time the user goes to open up the benchmark menu.

- Args:
  - None
- Keyword Args:
  - None

exit_benchmark

This procedure is in charge of switching back from the benchmark UI to Pycraft and making sure that the benchmark engine is reset so it behaves as expected next time the user goes to open up the benchmark menu.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

3.5.2 start_benchmark

This class is in charge of creating and setting up the different environments used for each component of the graphics benchmark. This is done so that they all start with the same starting conditions to make the tests fair for comparison between different stages of the benchmark, and also between different devices running this benchmark.

- Args:
  - None
- Keyword Args:
  - None

generate_benchmark

This function does the bulk of the setup that you would find between different areas of the graphics benchmark to make sure that each test is repeatable and setup in the same way.

- Args:
  - None
- Keyword Args:
  - create_display (bool): This option controls wether a Pygame surface object should be created or not. Often if a Pygame surface object isn't created here then it will be in the 'generate_opengl_benchmark' function which is below this.
- Output:
  - set_fps (array): This is an array of integers that stores FPS targets for the benchmark section of Pycraft, with each element being a different FPS to try to reach, getting progressively harder. The FPS from this array is updated every 500 iterations of the benchmark.
  - set_fps_length (int): This is the length of the 'set_fps' array, we use this instead of specifying an integer in order to allow us to make changes later on in Pycraft's development about how many targets to use for the benchmark section.
  - display (Pygame Surface | None): The display object is used throughout Pycraft. This is the identifier we use when we want to interact with/draw to/update Pycraft's gui. Pygame is the main windowing engine used in Pycraft. OR If the keyword parameter 'create_display' is set to False, then None is returned.
  - iteration (int): In the benchmarking process, iteration is used to keep track of how long the benchmark has been running.
  - fps_counter (int): This is used to store the index used to calculate the next element in the 'set_fps' array, this is used so Pycraft know's what to set the FPS to next, and what to set the caption to so that it displays the current FPS being tested.
  - max_iteration (int): This is used to calculate after how many iterations we move onto the next targeted FPS, currently this is set to increase the FPS every 500 'iteration's.

generate_opengl_benchmark

This function handles the specific setup for any OpenGL benchmark environment. This is still used in partnership with 'generate_benchmark' however does extend its functionality with OpenGL specific data.

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - display (Pygame Surface | None): The display object is used throughout Pycraft. This is the identifier we use when we want to interact with/draw to/update Pycraft's gui. Pygame is the main windowing engine used in Pycraft. OR If the keyword parameter 'create_display' is set to False, then None is returned.
  - ctx (Context object): This is used by ModernGL for loading OpenGL resources and enabling access to OpenGL features.
  - wnd (BaseWindow): This is used by ModernGL_window as the display object to use for rendering and additional resource loading.

3.5.3 clear_benchmark

This class is in charge of running a simple spacer to act as a gap between each of the graphics benchmarks. This is used as a time to reset arguments between each test, although that is not handled here.

- Args:
  - None
- Keyword Args:
  - None

run_spacer

This procedure is in charge of running a simple spacer to act as a gap between each of the graphics benchmarks. This is used as a time to reset arguments between each test, although that is not handled here.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
  - display (Pygame Surface): The display object is used throughout Pycraft. This is the identifier we use when we want to interact with/draw to/update Pycraft's gui. Pygame is the main windowing engine used in Pycraft.
  - background_color (array): An array containing the RGB colour values used to represent the colour of the background to the window at this time.
  - clock (Clock): The clock object is used by Pygame as a way of controlling the frame-rate and other frame-rate specific functions. We use this to limit the FPS throughout Pycraft.
- Keyword Args:
  - None

- Output:
  - None

3.6 blank_window_benchmark

3.6.1 run_blank_window_benchmark

This class is in charge of the blank window benchmark seen in the benchmark section of Pycraft.

- Args:
  - None
- Keyword Args:
  - None

start

This subroutine is used to render the blank window benchmark, accessible when run through the benchmark section of Pycraft. This test is one of three designed to stress your system. This one is usually considered the 'baseline' as it is the easiest to run.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
  - iteration (int): In the benchmarking process, iteration is used to keep track of how long the benchmark has been running
  - set_fps_length (int): This is the length of the 'set_fps' array, we use this instead of specifying an integer in order to allow us to make changes later on in Pycraft's development about how many targets to use for the benchmark section.
  - set_fps (array): This is an array of integers that stores FPS targets for the benchmark section of Pycraft, with each element being a different FPS to try to reach, getting progressively harder. The FPS from this array is updated every 500 iterations of the benchmark.
  - fps_counter (int): This is used to store the index used to calculate the next element in the 'set_fps' array, this is used so Pycraft know's what to set the FPS to next, and what to set the caption to so that it displays the current FPS being tested.
  - max_iteration (int): This is used to calculate after how many iterations we move onto the next targeted FPS, currently this is set to increase the FPS every 500 'iteration's.
  - display (Pygame Surface): The display object is used throughout Pycraft. This is the identifier we use when we want to interact with/draw to/update Pycraft's gui. Pygame is the main windowing engine used in Pycraft.
- Keyword Args:
  - None
- Output:
  - fps_list_X (array): Used to store the iteration of the benchmark. This correlates to a point, with this making up the X coordinate and 'fps_list_Y' making up the Y coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.

  - fps_list_Y (array): Used to store the FPS at a given iteration of the benchmark. This correlates to a point, with this making up the Y coordinate and 'fps_list_X' making up the X coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.

3.7 button_utils

3.7.1 draw_setting_elements

This class is in charge of rendering the button element that you can see used in the settings menu for Pycraft. Please note that the use of 'self' in this module is purely as a way to make changes to variables in Pycraft, and is not/should not be used in any other way for simplicity.

- Args:
  - None
- Keyword Args:
  - None

draw_multi_buttons

This function is in charge of rendering the multi-button element (where multiple options can be selected) that you can see used in the settings menu for Pycraft. Please note that the use of 'self' in this module is purely as a way to make changes to variables in Pycraft, and is not/should not be used in any other way for simplicity.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
  - button_pos (int): This is used to store the Y coordinate of the buttons position onscreen, so that elements can be rendered in the correct place.
  - button_text_array (array): This array stores a sequence of 1 or more strings that will be used to title each button.
  - font (Pygame Font): This parameter stores the font that will be used by default to render any text supplied to the function.
  - backup_font (Pygame Font): This parameter stores the font that will be used to render text supplied to this function when the default font can't be used. (likely due to limited support for some characters).
  - argument_variable (str): This parameter represents the dictionary key for the variable that this function modifies, for example, if the variable using self was used it may look like 'self.variable_name', then this parameter would be 'variable_name'.
  - hovering (bool): This parameter is used so that the setting menu knows when the user is hovering over a setting.
  - mouse_over (bool): This parameter is used to tell the settings menu when to display information messages for an option (often then the user's mouse is hovering over a setting, although the functionality for this is different to the 'hovering' parameter).
  - scrollbar_needed (bool): This parameter controls when the graphic this function draws should be offset to allow for a scroll-bar to be rendered.
  - aa (bool): This controls wether anti-aliasing should be used when rendering font.

- font_color (tuple): This tuple controls which colour the font should be rendered with and is also the colour that the graphic will turn when it is currently being overed over. This is adjustable through the theme selection menu.
- display (Pygame Surface): The display object is used throughout Pycraft. This is the identifier we use when we want to interact with/draw to/update Pycraft's gui. Pygame is the main windowing engine used in Pycraft.
- mouse_x (int): This stores the current X position of the user's cursor relative to the top-left corner of the window
- mouse_y (int): This stores the current Y position of the user's cursor relative to the top-left corner of the window
- accent_color (tuple): This tuple controls which colour the graphic should be rendered with, when that option is enabled.
- shape_color (tuple): This tuple controls which colour the graphic should be rendered with, by default.
- platform (str): This string tells the subroutine which operating system we are using. This is needed for OS specific operations.
- base_folder (str): This string is a file path to the resources for Pycraft on your device.
- use_mouse_input (bool): This parameter tells the function wether the user is using a keyboard and mouse, or a controller to interact with the element onscreen, as this can change how the element should detect events.
- sound (bool): This input controls wether the element onscreen should play a sound when it is interacted with.
- theme (str): This parameter represents what theme the user has currently selected.
- sound_volume (float): This parameter controls the volume at which should should be played at when the element is interacted with.
- settings_preset (str): This parameter controls which pre-set settings the user has chosen to use, this can be either 'low', 'medium', 'high' or 'adaptive', where each option focuses on either performance or visual quality in game.
- themeArray (array): This parameter stores all of the information needed to adjust the theme the user has selected, including 'font_color', 'background_color', 'shape_color', 'accent_color', 'secondary_font_color' for each of the 3 available themes.
- background_color (array): This array stores the RGB colour value used to represent the background colour of the element (this should be the same as the background colour to the rest of the window)
- secondary_font_color (tuple): This parameter stores the second font colour that can be used to add greater effect to a widget, for example better showing when an option is disabled.
- fps (float): This is the frame rate the game is targeted to try and run at. This is not a guaranteed value and should represent the maximum frame rate the game should be allowed to run at.
- render_fog (bool): This controls the rendering of fog effects in game, disabling this setting can improve performance, but lower visual quality.
- fancy_graphics (bool): This controls the rendering of more complex visual effects that serve only to look good, so that the user can control performance or visual quality.
- fancy_particles (bool): This controls the rendering of higher quality particles in game, so that the user can control performance or visual quality.
- average_fps (float): This stores the cumulative achieved frame rate from the last 1000 game cycles. This can be used then to calculate an average frame rate.

  - iteration (int): This counter is used to count up to 1000, and is used to calculate an average frame rate for those 1000 samples. Then this counter gets reset to 1 (to avoid ZeroDivisionError).
  - mouse_button_down (bool): This parameter controls when the user has opted to select an option, and although this can be remapped, it often represents when the user clicks on the onscreen element.
  - language (str): This procedure contains a list of all the supported languages Pycraft can be translated into.
  - logging_dictionary (dict): This dictionary is used to tell this subroutine if information messages are to be logged, this can be adjusted in settings.
  - output_log (bool): This option tells the subroutine if logged messages should also be outputted to the console.
  - translated_text (dict): This dictionary stores all the text that has been previously translated (like a cache). This improves performance and reduces the number of calls to external language servers (google translate). All text that is to be translated must first check this dictionary!
  - connection_permission (bool): This parameter controls wether Pycraft is allowed to connect to the internet. This can then be used to control a range of features, for example text translations and checking for updates to Pycraft.

- Keyword Args:
  - None
- Output:
  - button_text_height + 20 (int): This output represents the vertical height of the onscreen element, so that the next setting can be rendered below it. We add 20 as a form of padding between setting options, and this represents 20 pixels.
  - hovering (bool): This parameter is used so that the setting menu knows when the user is hovering over a setting.
  - mouse_over (bool): This parameter is used to tell the settings menu when to display information messages for an option (often then the user's mouse is hovering over a setting, although the functionality for this is different to the 'hovering' parameter).
  - fps (float): This is the frame rate the game is targeted to try and run at. This is not a guaranteed value and should represent the maximum frame rate the game should be allowed to run at.
  - aa (bool): This controls wether anti-aliasing should be used when rendering font.
  - render_fog (bool): This controls the rendering of fog effects in game, disabling this setting can improve performance, but lower visual quality.
  - fancy_graphics (bool): This controls the rendering of more complex visual effects that serve only to look good, so that the user can control performance or visual quality.
  - fancy_particles (bool): This controls the rendering of higher quality particles in game, so that the user can control performance or visual quality.
  - average_fps (float): This stores the cumulative achieved frame rate from the last 1000 game cycles. This can be used then to calculate an average frame rate.
  - iteration (int): This counter is used to count up to 1000, and is used to calculate an average frame rate for those 1000 samples. Then this counter gets reset to 1 (to avoid ZeroDivisionError).
  - themeArray (array): This parameter stores all of the information needed to adjust the theme the user has selected, including 'font_color', 'background_color', 'shape_color', 'accent_color', 'secondary_font_color' for each of the 3 available themes.

  - font_color (tuple): This tuple controls which colour the font should be rendered with and is also the colour that the graphic will turn when it is currently being overed over. This is adjustable through the theme selection menu.
  - background_color (array): This array stores the RGB colour value used to represent the background colour of the element (this should be the same as the background colour to the rest of the window)
  - shape_color (tuple): This tuple controls which colour the graphic should be rendered with, by default.
  - accent_color (tuple): This tuple controls which colour the graphic should be rendered with, when that option is enabled.
  - secondary_font_color (tuple): This parameter stores the second font colour that can be used to add greater effect to a widget, for example better showing when an option is disabled.
  - theme (str): This parameter represents what theme the user has currently selected.
  - mouse_button_down (bool): This parameter controls when the user has opted to select an option, and although this can be remapped, it often represents when the user clicks on the onscreen element.
  - translated_text (dict): This dictionary stores all the text that has been previously translated (like a cache). This improves performance and reduces the number of calls to external language servers (google translate). All text that is to be translated must first check this dictionary!

draw_buttons

This function is in charge of rendering the chained button element (where only one option can be chosen) that you can see used in the settings menu for Pycraft. Please note that the use of 'self' in this module is purely as a way to make changes to variables in Pycraft, and is not/should not be used in any other way for simplicity.

- Args:
  - button_pos ():
  - self ():
  - button_text_array ():
  - font ():
  - value ():
  - backup_font ():
  - argument_variable ():
  - hovering ():
  - mouse_over ():
  - files_to_remove ():
  - clear_languages ():
  - scanned_files ():
  - scrollbar_needed ():
  - font_color ():
  - aa ():
  - display ():
  - mouse_x ():

  - mouse_y ():
  - sound ():
  - accent_color ():
  - shape_color ():
  - platform ():
  - base_folder ():
  - remove_file_permission
  - settings_preset ():
  - fps ():
  - render_fog ():
  - fancy_graphics ():
  - fancy_particles ():
  - average_fps ():
  - iteration ():
  - use_mouse_input ():
  - sound_volume ():
  - themeArray ():
  - background_color ():
  - secondary_font_color ():
  - language ():
  - logging_dictionary ():
  - output_log ():
  - translated_text ():
  - connection_permission ():
- Keyword Args:
  - None
- Output:
  - button_text_height + 20 ():
  - hovering ():
  - mouse_over ():
  - fps ():
  - aa ():
  - render_fog ():
  - fancy_graphics ():
  - fancy_particles ():
  - average_fps ():

  - iteration ():
  - themeArray ():
  - font_color ():
  - background_color ():
  - shape_color ():
  - accent_color ():
  - secondary_font_color ():
  - translated_text ():

3.8 camera_utils

3.8.1 compute_camera

NYI

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - None

camera_move_state

NYI

- Args:
  - camera ():
  - direction ():
  - activate ():
  - RIGHT ():
  - LEFT ():
  - FORWARD ():
  - BACKWARD ():
  - UP ():
  - DOWN ():
  - STILL ():
  - POSITIVE ():
  - NEGATIVE ():
- Keyword Args:

  - None
- Output:
  - None

get_camera_values

NYI

- Args:
  - self ():
  - camera ():
  - camera_up ():
  - compile_math ():
  - STILL ():
  - POSITIVE ():
  - NEGATIVE ():
- Keyword Args:
  - None
- Output:
  - cam_matrix ():
  - position ():

compute_camera_dir

NYI

- Args:
  - self ():
  - camera ():
  - POSITIVE ():
  - NEGATIVE ():
- Keyword Args:
  - None
- Output:
  - None

3.9 caption_utils

3.9.1 generate_captions

NYI

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - None

get_loading_caption

NYI

- Args:
  - version ():
  - num ():
- Keyword Args:
  - None
- Output:
  - None

get_normal_caption

NYI

- Args:
  - location ():
  - detailed_captions ():
  - play_time ():
  - x ():
  - y ():
  - z ():
  - total_move_x ():
  - total_move_y ():
  - total_move_z ():
  - fps_overclock ():
  - current_fps ():
  - iteration ():

  - version ():
  - current_memory_usage ():
  - theme ():
  - fps ():
  - average_fp ():
- Keyword Args:
  - None
- Output:
  - None

set_OpenGL_caption

NYI

- Args:
  - self ():
  - play_time ():
  - Time_Percent ():
  - day ():
  - total_move_x ():
  - total_move_y ():
  - total_move_z ():
  - weather ():
- Keyword Args:
  - None
- Output:
  - None

3.10 character_designer

3.10.1 generate_character_designer

This class is in charge of rendering the character customiser GUI, currently this is just a black window as we need to get the game engine right and work on a character model, in later versions the character will hopefully be rendered in 3D and the result (as well as any changes), should be able to be viewed live.

- Args:
  - None
- Keyword Args:
  - None

character_designer_gui

This subroutine does the bulk of the rendering for the character customiser GUI, currently this is just a black window as we need to get the game engine right and work on a character model, in later versions the character will hopefully be rendered in 3D and the result (as well as any changes), should be able to be viewed live.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

3.11 credits

3.11.1 generate_credits

This class is in charge of creating the credits menu from a file called 'credits_config.json' in the 'data files' folder.

- Args:
  - None
- Keyword Args:
  - None

credits_gui

This subroutine is in charge of creating the credits menu from a file called 'credits_config.json' in the 'data files' folder. Every key has a purpose, with the '<spacerN>' syntax being used to specify any spaces or breaks to make the credits menu easier to read. (The 'N' being used to count the number of spaces as without this the spacer gets ignored.)

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - None

3.12 custom_theme_utils

3.12.1 draw_setting_elements

draw_custom_theme_options

3.13 directory_utils

3.13.1 draw_setting_elements

draw_directory_structure

3.14 display_utils

3.14.1 display_functionality

This class is in charge of handling calls to subroutines, as well as enabling basic GUI functionality to Pycraft, this is called by many GUIs and is heavily customisable. This is designed to simplify GUI design and make it easier to roll out some changes to every GUI in Pycraft.

- Args:
  - None
- Keyword Args:
  - None

core_display_functions

This subroutine is in charge of the basic functionality you would expect from Pycraft's GUIs, it handles events, mouse and controller positions, as well as allowing for great customisability, meaning its designed to be called by most GUI's and be flexible enough to be functional.

- Args:
  - platform (str): This string tells the subroutine which operating system we are using. This is needed for OS specific operations.
  - base_folder (str): This string is a file path to the resources for Pycraft on your device.
  - display (Pygame Surface): The display object is used throughout Pycraft. This is the identifier we use when we want to interact with/draw to/update Pycraft's gui. Pygame is the main windowing engine used in Pycraft.
  - use_mouse_input (bool):
  - average_fps (float):
  - iteration (int):
  - mouse_x (int):
  - mouse_y (int):
  - x_scale_factor (float):

- y_scale_factor (float):
- go_to (str):
- joystick_exit (bool):
- joystick_hat_pressed (bool):
- window_in_focus (bool):
- saved_window_width (int):
- saved_window_height (int):
- clock (Pygame Clock):
- sound (bool):
- input_key (dict):
- input_configuration (dict):
- extended_developer_options (bool):
- logging_dictionary (dict): This dictionary is used to tell this subroutine if information messages are to be logged, this can be adjusted in settings.
- output_log (bool): This option tells the subroutine if logged messages should also be outputted to the console.
- vsync (bool):
- window_icon (Pygame Surface): This is the icon we use in the caption (and in the taskbar on some supported OS') for Pycraft.
- sound_volume (float):
- variable_data (dict):
- version (str)
- Pycraft's current version.
- background_color (tuple):
- font_color (tuple):
- fullscreen (bool):
- startup_animation (bool):
- run_timer (float):
- data_average_fps (arr):
- data_CPU_usage (arr):
- data_current_fps (arr):
- data_memory_usage (arr):
- timer (float):
- data_average_fps_Max (float):
- data_CPU_usage_Max (float):
- data_current_fps_Max (float):
- data_memory_usage_Max (float):

  - joystick_zoom (str):
  - mouse_button_down (bool):
  - error_message (str):
  - error_message_detailed (str):
- Keyword Args:
  - location="home" (str):
  - checkEvents=True (bool):
  - resize=True (bool):
  - return_events=False (bool):
  - disable_events=False (bool):
- Output:
  - displayEvents (arr):
  - display (Pygame Surface): The display object is used throughout Pycraft. This is the identifier we use when we want to interact with/draw to/update Pycraft's gui. Pygame is the main windowing engine used in Pycraft.
  - mouse_button_down (bool):
  - go_to (str):
  - startup_animation (bool):
  - run_timer (float):
  - current_fps (float):
  - average_fps (float):
  - iteration (int):
  - saved_window_width (int):
  - saved_window_height (int):
  - window_in_focus (bool):
  - joystick_exit (bool):
  - x_scale_factor (float):
  - y_scale_factor (float):
  - real_window_width (int):
  - real_window_height (int):
  - mouse_x (int):
  - mouse_y (int):
  - data_average_fps (arr):
  - data_CPU_usage (arr):
  - data_current_fps (arr):
  - data_memory_usage (arr):
  - timer (float):

- data_average_fps_Max (float):
- data_CPU_usage_Max (float):
- data_current_fps_Max (float):
- data_memory_usage_Max (float):
- joystick_zoom (str):
- clock (Pygame Clock):
- joystick_hat_pressed (bool):
- fullscreen (bool):
- joystick_connected (bool):

3.14.2 display_utils

update_display

set_display

generate_min_display

get_display_location

get_play_status

3.14.3 display_animations

fade_in

fade_out

3.15 drawing_utils

3.15.1 draw_rose

create_rose

3.15.2 generate_graph

create_devmode_graph

3.16 drawing_window_benchmark

3.16.1 run_drawing_window_benchmark

This class is in charge of the drawing window benchmark seen in the benchmark section of Pycraft.

- Args:

  - None
- Keyword Args:
  - None

start

This subroutine is used to render the drawing window benchmark, accessible when run through the benchmark section of Pycraft. This test is the second of three designed to stress your system. This one is usually used to measure the performance of CPU rendering with Pygame on your device.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
  - iteration (int): In the benchmarking process, iteration is used to keep track of how long the benchmark has been running
  - Setfpslength (int): This is the length of the 'Setfps' array, we use this instead of specifying an integer in order to allow us to make changes later on in Pycraft's development about how many targets to use for the benchmark section.
  - Setfps (array): This is an array of integers that stores FPS targets for the benchmark section of Pycraft, with each element being a different FPS to try to reach, getting progressively harder. The FPS from this array is updated every 500 iterations of the benchmark.
  - fpscounter (int): This is used to store the index used to calculate the next element in the 'Setfps' array, this is used so Pycraft know's what to set the FPS to next, and what to set the caption to so that it displays the current FPS being tested.
  - Maxiteration (int): This is used to calculate after how many iterations we move onto the next targeted FPS, currently this is set to increase the FPS every 500 'iteration's.
  - display (Pygame Surface): The display object is used throughout Pycraft. This is the identifier we use when we want to interact with/draw to/update Pycraft's gui. Pygame is the main windowing engine used in Pycraft.
- Keyword Args:
  - None
- Output:
  - fpslistX (array): Used to store the iteration of the benchmark. This correlates to a point, with this making up the X coordinate and 'fpslistY' making up the Y coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.
  - fpslistY (array): Used to store the FPS at a given iteration of the benchmark. This correlates to a point, with this making up the Y coordinate and 'fpslistX' making up the X coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.

3.17 dropdown_utils

3.17.1 draw_setting_elements

draw_dropdown

3.18 error_utils

3.18.1 generate_error_screen

error_screen

3.19 extended_benchmark

3.19.1 Loadbenchmark

This class is in charge of loading and running the 3 window benchmarks in the correct order and returning the results of each run back to the benchmark GUI for processing.

- Args:
  - None
- Keyword Args:
  - None

run

This subroutine is in charge of loading and running the 3 window benchmarks in the correct order and returning the results of each run back to the benchmark GUI for processing.

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - fps_list_x_1 (array): Used to store the iteration of the benchmark. This correlates to a point, with this making up the X coordinate and 'fps_list_y_1' making up the Y coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.
  - fps_list_y_1 (array): Used to store the FPS at a given iteration of the benchmark. This correlates to a point, with this making up the Y coordinate and 'fps_list_x_1' making up the X coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.
  - fps_list_x_2 (array): Used to store the iteration of the benchmark. This correlates to a point, with this making up the X coordinate and 'fps_list_y_2' making up the Y coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.

  - fps_list_y_2 (array): Used to store the FPS at a given iteration of the benchmark. This correlates to a point, with this making up the Y coordinate and 'fps_list_x_2' making up the X coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.
  - fps_list_x_3 (array): Used to store the iteration of the benchmark. This correlates to a point, with this making up the X coordinate and 'fps_list_y_1' making up the Y coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.
  - fps_list_y_1 (array): Used to store the FPS at a given iteration of the benchmark. This correlates to a point, with this making up the Y coordinate and 'fps_list_x_3' making up the X coordinate. These points are later plotted (after a bit of processing) in the benchmark results screen on a line graph.

3.20 file_utils

3.20.1 delete_files

clear_temporary_files

3.20.2 scan_folder

search_files

3.20.3 fix_installer

set_install_location

get_install_location

3.20.4 pycraft_config_utils

read_input_key

read_main_save

repair_lost_save

save_pycraft_config

3.21 game_engine

3.21.1 create_game_engine

This class is responsible for the setup, loading and running of the game engine.

- Args:
  - None
- Keyword Args:
  - None

start

This subroutine is responsible for telling ModernGL and ModernGL_window that our Pygame display is to be rendered to.

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - ctx (Context object): This is used by ModernGL for loading OpenGL resources and enabling access to OpenGL features.
  - wnd (BaseWindow): This is used by ModernGL_window as the display object to use for rendering and additional resource loading.

game_engine

This subroutine is responsible for loading and running Pycraft's game engine.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

3.22 home

3.22.1 generate_home

This class is in charge of loading the resources used in the main menu of Pycraft. (Alternatively also called the 'title screen' or 'home screen') This class is also in charge of rendering the main menu.

- Args:
  - None
- Keyword Args:
  - None

create_banner

This subroutine is used to render the messages the user may see at the bottom of the home screen. This is run in parallel (thread).

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

home_gui

This subroutine is in charge of loading the resources used in the main menu of Pycraft. (Alternatively also called the 'title screen' or 'home screen') This subroutine is also in charge of rendering the main menu.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

3.23 image_utils

3.23.1 convert_image

pil_image_to_surface

surface_to_pil_image

3.23.2 pygame_image_extensions

display_to_string

3.23.3 transparency_effects

create_background_image

3.23.4 tkinter_installer

open_img

3.24 input_utility

3.24.1 identify_hex

3.24.2 identify_text

3.24.3 identify_rgb

3.25 install

3.25.1 begin_install

install_screen_one

button_check

install_screen_two

get_dir

install_screen_three

render_progress_bar

install_screen_four

desktop_is_checked

start_is_checked

toggle_release_notes

on_exit

3.26 installer_home

3.26.1 installer_home

start

get_version

3.27 installer_main

3.27.1 run_installer

__init__

Initialize

3.28 installer_utils

3.28.1 get_installer_data

get_data

3.28.2 core_installer_functionality

close

home

outdated_detector

3.28.3 file_manipulation

move_files

download_and_install

search_files

remove_files

3.29 integrated_installer_utils

3.29.1 integrated_installer

This class is in charge of connecting the installer and Pycraft together. This class is used to check for updates to Pycraft.

- Args:
  - None
- Keyword Args:
  - None

check_versions

This subroutine runs the command 'pip list –outdated' in a thread. The results of this command are then processed to check of Pycraft is outdated. This is run in parallel (thread).

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

3.29.2 check_connection

This class is used to check if your PC has a working network connection so that updates can be checked for successfully.

- Args:
  - None
- Keyword Args:
  - None

test

This subroutine attempts to ping google's servers. If this is successfully it means there is currently an internet connection to your PC. This is not run in a thread, so therefore there is a 1 second timeout. This means that if there isn't an internet connection available it doesn't slow down Pycraft's startup too much.

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - (bool): This subroutine will return True if an internet connection can be established. If an internet connection could not be established, nothing is returned.

3.30 inventory

3.30.1 generate_inventory

This class is in charge of setting up, managing, loading and running the inventory GUI. This GUI gets loaded as a separate process at the start of the game engine and runs only when the game engine sends a command. This is run in parallel (process).

- Args:

  - None
- Keyword Args:
  - None

__init__

This class is in charge of turning the dictionary the user enters as a parameter (that contains all the data required to properly display the inventory GUI) into a 'generate_inventory' object in a similar style to the 'pycraft_main' program. This also initializes the inventory's required modules.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
  - dictionary (dict): This parameter holds all the data that is stored as 'self' in the game engine. This includes the configuration and user defined settings. Note that the input to this variable must be sorted first to remove any modules or module specific objects. For example a (Pygame Surface) type object as that will cause errors when creating the new process.
- Keyword Args:
  - None
- Output:
  - None

inventory_gui

This subroutine is in charge of loading and running the inventory GUI. This GUI gets loaded as a separate process at the start of the game engine and runs only when the game engine sends a command. This is run in parallel (process).

- Args:
  - dictionary (dict): This parameter holds all the data that is stored as 'self' in the game engine. This includes the configuration and user defined settings. Note that the input to this variable must be sorted first to remove any modules or module specific objects. For example a (Pygame Surface) type object as that will cause errors when creating the new process.
  - start_inventory (Multiprocessing Event object): This parameter is an event object used to tell the inventory when the game engine wants to have the inventory displayed. When this event is not set the inventory will wait.
- Keyword Args:
  - None
- Output:
  - None

3.31 loading_screen

3.31.1 generate_load_screen

This class is in charge of setting up, managing, loading and running the loading GUI. This GUI gets loaded as a separate process at the start of the game engine and runs whilst the game engine loads to indicate that Pycraft hasn't crashed and how far through loading the game engine we are. This is run in parallel (process).

- Args:
  - None
- Keyword Args:
  - None

__init__

This class is in charge of turning the dictionary the user enters as a parameter (that contains all the data required to properly display the loading screen GUI) into a 'generate_load_screen' object in a similar style to the 'pycraft_main' program. This also initializes the load screen's required modules.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
  - dictionary (dict): This parameter holds all the data that is stored as 'self' in the game engine. This includes the configuration and user defined settings. Note that the input to this variable must be sorted first to remove any modules or module specific objects. For example a (Pygame Surface) type object as that will cause errors when creating the new process.
- Keyword Args:
  - None
- Output:
  - None

load

This subroutine is in charge of loading and running the loading GUI. This GUI gets loaded as a separate process at the start of the game engine and runs whilst the game engine loads to indicate that Pycraft hasn't crashed and how far through loading the game engine we are. This is run in parallel (process).

- Args:
  - dictionary (dict): This parameter holds all the data that is stored as 'self' in the game engine. This includes the configuration and user defined settings. Note that the input to this variable must be sorted first to remove any modules or module specific objects. For example a (Pygame Surface) type object as that will cause errors when creating the new process.
  - start_loading (Multiprocessing Event object): This parameter is an event object used to tell the load screen when the game engine starts to load. When this event is not set the load screen will wait to be called again.
- Keyword Args:
  - None

- Output:
  - None

3.32 logging_utils

3.32.1 create_log_message

This class adds logging support to Pycraft.

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - None

update_log_information

This subroutine handles the formatting, output and logging of all non-critical information. This can be a handy debugging tool.

- Args:
  - logging_dictionary (dict): This dictionary is used to tell this subroutine if information messages are to be logged, this can be adjusted in settings.
  - text (str): This string contains the piece of information to log.
  - output_log (bool): This option tells the subroutine if logged messages should also be outputted to the console.
  - platform (str): This string tells the subroutine which operating system we are using. This is needed for OS specific operations.
  - base_folder (str): This string is a file path to the resources for Pycraft on your device.
- Keyword Args:
  - None
- Output:
  - None

update_log_warning

This subroutine handles the formatting, output and logging of all non-critical warnings that could cause errors if not dealt with during the execution of Pycraft. This can be a handy debugging tool.

- Args:
  - logging_dictionary (dict): This dictionary is used to tell this subroutine if warning messages are to be logged, this can be adjusted in settings.
  - text (str): This string contains the piece of information to log.
  - output_log (bool): This option tells the subroutine if logged messages should also be outputted to the console.
  - platform (str): This string tells the subroutine which operating system we are using. This is needed for OS specific operations.
  - base_folder (str): This string is a file path to the resources for Pycraft on your device.
- Keyword Args:
  - None
- Output:
  - None

update_log_error

This subroutine handles the formatting, output and logging of all critical errors in Pycraft. These must be dealt with immediately and will stop the execution of Pycraft, or could cause some things to not behave as expected.

- Args:
  - logging_dictionary (dict): This dictionary is used to tell this subroutine if error messages are to be logged, this can be adjusted in settings.
  - text (str): This string contains the piece of information to log.
  - output_log (bool): This option tells the subroutine if logged messages should also be outputted to the console.
  - platform (str): This string tells the subroutine which operating system we are using. This is needed for OS specific operations.
  - base_folder (str): This string is a file path to the resources for Pycraft on your device.
- Keyword Args:
  - None
- Output:
  - None

3.32.2 log_file

This class handles the writing to and formatting of the log file.

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - None

clear_log

This subroutine clears the log file. This is often called at startup to prevent the log file becoming too long.

- Args:
  - platform (str): This string tells the subroutine which operating system we are using. This is needed for OS specific operations.
  - base_folder (str): This string is a file path to the resources for Pycraft on your device.
- Keyword Args:
  - None
- Output:
  - None

update_log

This subroutine updates the log file by appending new information to the end. This is usually called every time a log is made.

- Args:
  - platform (str): This string tells the subroutine which operating system we are using. This is needed for OS specific operations.
  - base_folder (str): This string is a file path to the resources for Pycraft on your device.
  - text (str): This string contains the formatted log which will be added to the log.
- Keyword Args:
  - None
- Output:
  - None

3.33 main

3.34 map_gui

3.34.1 generate_map_gui

This class is in charge of setting up, managing, loading and running the map GUI. This GUI gets loaded as a separate process at the start of the game engine and runs only when the game engine sends a command. This is run in parallel (process).

- Args:
  - None
- Keyword Args:
  - None

__init__

This class is in charge of turning the dictionary the user enters as a parameter (that contains all the data required to properly display the map GUI) into a 'generate_map_gui' object in a similar style to the 'pycraft_main' program. This also initializes the map's required modules.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
  - dictionary (dict): This parameter holds all the data that is stored as 'self' in the game engine. This includes the configuration and user defined settings. Note that the input to this variable must be sorted first to remove any modules or module specific objects. For example a (Pygame Surface) type object as that will cause errors when creating the new process.
- Keyword Args:
  - None
- Output:
  - None

get_map_pos

This maths based subroutine is used to calculate the position the map should be rendered onscreen.

- Args:
  - in_x (float): This parameter represents the user's position in game on the X axis.
  - in_z (float): This parameter represents the user's position in game on the Z axis.
- Keyword Args:
  - None
- Output:
  - x (int): The X coordinate to render the map.

  - z (int): The Y coordinate to render the map. (In 2D space there is no 'Z' or depth axis, therefore because we don't care about the user's height position here, we use their Z position in 3D space to represent their Y coordinate in 2D space.)

map_gui

This subroutine is in charge of loading and running the map GUI. This GUI gets loaded as a separate process at the start of the game engine and runs only when the game engine sends a command. This is run in parallel (process).

- Args:
  - dictionary (dict): This parameter holds all the data that is stored as 'self' in the game engine. This includes the configuration and user defined settings. Note that the input to this variable must be sorted first to remove any modules or module specific objects. For example a (Pygame Surface) type object as that will cause errors when creating the new process.
  - start_inventory (Multiprocessing Event object): This parameter is an event object used to tell the inventory when the game engine wants to have the inventory displayed. When this event is not set the map will wait.
- Keyword Args:
  - None
- Output:
  - None

3.35 math_utils

3.35.1 math_functions

gl_look_at

normalize

compute_position

perspective_fov

look_at

multiply

3.35.2 compiled_math_functions

gl_look_at

compute_position

normalize

perspective_fov

look_at

multiply

3.36 menu_utils

3.36.1 access_other_guis

access_gui

3.37 particle_utils

3.37.1 particles

emit_gpu

gen_particles

projection

3.38 pycraft_main

3.38.1 startup

This class is used to make sure Pycraft starts up and initializes properly.

- Args:
  - None
- Keyword Args:
  - None

__init__

This class initializes Pycraft, this class also creates the 'self' dictionary used throughout Pycraft.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

3.38.2 Initialize

This class is used to start Pycraft. It is also responsible for 'connecting' you to different menus and making sure Pycraft starts up correctly.

- Args:
  - None
- Keyword Args:
  - None

menu_selector

This subroutine is used to transfer you between different GUIs and programs used in Pycraft. This is also used to make sure that different menus in Pycraft get everything they need in order to start properly.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

start

This subroutine is used as the default start-up option for Pycraft. This will initialize a display, create all the variables (by using an earlier subroutine in this module) and get Pycraft ready for handing over to the main menu (home). Calling this subroutine starts Pycraft.

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - None

QueryVersion

This subroutine can be used to return the current version of Pycraft. Version Naming Pycraft's versions will always now follow the structure; "vA.B.C" * Where "A" is the major revision number. * Where "B" is the minor revision number. * Where "C" is the patch and developer preview numbers (combined). Every version of Pycraft as of the 27/10/2022 (DD/MM/YYYY) must feature all 3 values. Updates also now go sequentially, so Pycraft v9.6.4 is newer than Pycraft v9.5.7. If either of the "A" or "B" version numbers is incremented in a release, documentation MUST be suitably updated, in addition Pycraft MUST be released on PyPi, SourceForge and as a release on GitHub.

- Args:
  - None

- Keyword Args:
  - None
- Output:
  - version (str): Pycraft's current version.

start

This subroutine is responsible for starting Pycraft, this can be used to call Pycraft externally, potentially as part of another program.

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - None

3.39 pycraft_startup_utils

3.39.1 startup_test

This class is in charge running checks on your hardware and Pycraft's file structure to make sure any problems with incorrect file paths are identified quickly and you get the best experience on your hardware.

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - None

test_for_resource

This subroutine checks a given file path to see if the resource it is expecting to find is present. If a resource is not at the required location then an error is returned.

- Args:
  - base_folder (str): This string is a file path to the resources for Pycraft on your device.
  - name (str): This is the name of the file we are trying to find.
  - path (str): This is the path from the 'pycraft' directory to where we are expecting the file to be.
- Keyword Args:
  - None
- Output:

  - error_message (str) OR None: If an error occurs whilst navigating to the required file path (potentially a folder may have moved) or the file we where expecting to find is not present then 'error_message' is returned. If no problems occur and the resource is found then 'None' gets returned.
  - error_message_detailed (str) OR None: If an error occurs whilst navigating to the required file path (potentially a folder may have moved) or the file we where expecting to find is not present then 'error_message_detailed' is returned. If no problems occur and the resource is found then 'None' gets returned. This output contains more details about exactly what error has occured and can be enabled
  - for testing or debug purposes usually
  - in the settings menu. This is a handy debugging tool.

resource_not_found

This subroutine creates the error message that gets returned if a resource is not at its expected location.

- Args:
  - name (str): This is the name of the file we are trying to find.
  - path (str): This is the path from the 'pycraft' directory to where we are expecting the file to be.
- Keyword Args:
  - None
- Output:
  - error_message (str): If an error occurs whilst navigating to the required file path (potentially a folder may have moved) or the file we where expecting to find is not present then 'error_message' is returned.
  - error_message_detailed (str): If an error occurs whilst navigating to the required file path (potentially a folder may have moved) or the file we where expecting to find is not present then 'error_message_detailed' is returned. This output contains more details about exactly what error has occured and can be enabled
  - for testing or debug purposes usually
  - in the settings menu. This is a handy debugging tool.

pycraft_self_test

This subroutine compares the minimum requirements of Pycraft to the specs of your hardware to see if we can run Pycraft on your PC. Specs:

- OpenGL v2.8 or newer (potentially needs to be reviewed).
- SDL v2 or newer.
- 260 MB of RAM or more (potentially need to be reviewed).
- Args:
  - window_icon (Pygame Surface): This is the icon we use in the caption (and in the taskbar on some supported OS') for Pycraft.
- Keyword Args:
  - None
- Output:
  - None

pycraft_resource_test

This subroutine is in charge of checking for every resource required by Pycraft to make sure that it is where Pycraft will expect it to be when it is required by other areas of the game. Any problems raised here may mean something is wrong with the structure of Pycraft. Problems here after an update or when you first install Pycraft can indicate an error with the install. This is run in parallel (thread).

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
  - override (bool): This is used to forcefully run 'pycraft_resource_test'. This is used to allow the user to check for problems in the settings menu (in the 'Storage and permissions' section).
- Keyword Args:
  - None
- Output:
  - None

3.40 registry_utils

3.40.1 generate_registry

This class is in charge of setting all the default values, and making sure every variable defined in 'self' exists, reducing the risk of errors because variables are not defined. This is sorted alphabetically.

- Args:
  - None
- Keyword Args:
  - None
- Output:
  - None

registry

This subroutine is in charge of setting all the default values, and making sure every variable defined in 'self' exists, reducing the risk of errors because variables are not defined. This is sorted alphabetically. This must always be called at startup (in 'pycraft_main').

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

3.41 remapping_utils

3.41.1 draw_setting_elements

draw_remap_function

3.42 save_menu

3.42.1 generate_save_menu

save_menu_gui

3.43 seasonal_events_utils

3.43.1 configure_seasonal_event

is_seasonal_event

3.44 setting_preset_utils

3.44.1 presets

update_profile

3.45 settings

3.45.1 generate_settings

This class is in charge of loading the structure for the settings menu, and rendering the settings menu properly.

- Args:
  - None
- Keyword Args:
  - None

restart_function

This subroutine adds restarting functionality into Pycraft. To do this we run a command in a separate process 'python main.py' which launches a separate instance of Pycraft before we then close the current instance.

- Args:
  - platform (str): This string tells the subroutine which operating system we are using. This is needed for OS specific operations.
  - base_folder (str): This string is a file path to the resources for Pycraft on your device.
- Keyword Args:

  - None
- Output:
  - None

settings_gui

This subroutine is in charge of rendering the settings menu and applying all changes to their corresponding variables throughout Pycraft.

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

3.46 settings_utils

3.46.1 draw_setting_elements

create_information_message

3.47 setup

3.48 shader_utils

3.48.1 load_programs

load_program_text

load_program_files

3.49 shadow_mapping_utils

3.49.1 shadowmapping_mathematics

compute_celestial_entities

compute_shadows

3.50 slider_utils

3.50.1 draw_setting_elements

draw_slider

3.51 sound_utils

3.51.1 play_sound

play_inventory_sound

play_click_sound

play_footsteps_sound

play_ambient_sound

play_thunder_sound

play_rain_sound

3.52 startup_animation

3.52.1 generate_startup_gui

This class is in charge of running the startup menu and also creating a thread that checks Pycraft's directory for required resources.

- Args:
  - None
- Keyword Args:
  - None

init

This subroutine is in charge of running the startup menu and also creating a thread that checks Pycraft's directory for required resources. This GUI runs before the theme decision menu, so if the user hasn't already set a theme then the startup animation will default to default. (The default theme for Pycraft is 'dark').

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None

- Output:
  - None

startup_gui

This subroutine is in charge of running the startup menu and also creating a thread that checks Pycraft's directory for required resources. This GUI runs before the theme decision menu, so if the user hasn't already set a theme then the startup animation will default to default. (The default theme for Pycraft is 'dark').

- Args:
  - self (dict): This is used by Pycraft as a way of storing it's current configuration and behaviour and is required by most GUIs. Its use should be reduced where possible for readability reasons.
- Keyword Args:
  - None
- Output:
  - None

3.53 text_utils

3.53.1 installer_text

create_text

3.53.2 text_formatter

format_text

3.53.3 generate_text

load_quick_text

3.53.4 text_wrap

blit_text

3.54 theme_gui

3.54.1 create_theme_selection_menu

get_theme_gui

3.55 theme_utils

3.55.1 determine_theme_colours

get_colors

3.56 threading_utils

3.56.1 pycraft_core_threads

general_threading_utility

adaptive_mode

3.57 tkinter_utils

3.57.1 tkinter_info

get_permissions

create_tkinter_window

3.57.2 tkinter_installer

create_display

3.58 toggle_utils

3.58.1 draw_setting_elements

draw_toggle

3.59 translation_utils

3.59.1 translation_caching

write_cache

read_cache

3.59.2 string_translator

change_language

3.60 uninstall

3.60.1 begin_uninstall

uninstall_screen_one

get_confirmation

remove_all

render_progress_bar

remove_but_keep_save

render_progress_bar

remove_but_leave

render_progress_bar

finish_uninstall

3.61 update

3.61.1 begin_update

update_screen_one

update_screen_two

update_options

render_progress_bar

finished_update

3.62 weather_utils

3.62.1 compute_weather

compute_cloud_model

generate_perlin_noise_2d

f

compute_cloud_noise

blend_weather

mix

compute_weather

CHAPTER

FOUR

FREQUENTLY ASKED QUESTIONS

4.1 Introduction

Welcome to Pycraft's new dedicated section to help with any questions and problems you may have about Pycraft. This is by no means a complete guide, and neither is it finished, however we hope that we can help you fix any problems and questions you may have. If we have missed anything, then this section is constantly being updated and adjusted, so let us know and we will happily make improvements based on your feedback! Please also note that, especially for the tutorial section, there will be dedicated tutorials for different platforms, currently we support Windows 10/11 and Linux (Ubuntu is our distro of choice for testing Pycraft on Linux) however if you have a problem on another platform, and wish to share how you fixed it, let us know and we will happily include that here as well for the convenience of others!

4.2 Frequently seen problems

Sometimes, when running Pycraft - usually immediately after installing it - you can occasionally run into problems, wether thats outdated versions of Python, Tkinter not being installed, or something isn't quite right with your drivers. Now sometimes, we will admit, the bug has more to do with Pycraft than anything else on your system, but over time we have been adapting and improving our debugging and testing regimes and this is becoming less of a significant factor. In 2021, we managed to comfortably half the amount of bugs we discovered in released versions of Pycraft, and did nearly that again in 2022. Now, this means that, whilst there is still more we could do - and are looking at doing - there are still sometimes problems that we cannot address through Pycraft. Some of these bugs can be related to platforms, and may only be present on Linux rather than Windows, and then only on some versions of Linux with some hardware configurations. Therefore these tutorials will try and explain the problem in detail, as well as what we believe the solution to be, and will occasionally discuss things we are planning on doing to address there problems although again, not all problems we can address through Pycraft. Now, these tutorials are designed to be as user friendly as possible, and include specific solutions to some problems that might vary based on your platform or Operating System.