

Implementation

Paul J. Schmiedtke

January 29, 2023

Contents

1	Integrated Development Environment	2
1.1	Design	2
1.2	Functionality	3
1.3	Loading addon in to KSP	8
2	Learning Application	9
2.1	Mission design	9
2.2	Parser - kRPC	11
3	Installer development	13
4	Extensibility	13
	References	16

The choice of Kerbal Space Program as the basis for the development of the learning application implies that needed features that the game does not have natively must be added. Because the game and its world motivate the player to explore. It features spacecraft-centric as well as orbit map views natively. Thus, a code editor and debug console for executed scripts as well as the ability to interact with the game using Python are missing. The development can be segmented into the independent modules *IDE* and *Learning Application*. The latter covers the development of the infrastructure to interface with KSP using Python as well as workflows and manuals to deploy, maintain and extend the software in section 2. The *IDE* simplifies using the learning application since it makes additional software obsolete. Its design is addressed in section 1.

1 Integrated Development Environment

The user interface is to be developed for the last evolution of KSP, which is Kerbal Space Program 1.12. While no new minor versions will be published, patches are released occasionally. The latest was released on November 2, 2022 [1]. This means that no extensive changes will be made to game which ensures a long term support for the developments. The game is made in the game engine Unity 2019.4.18f1 [2]. While the UI could be made without using the Unity Editor, it provides the ability to assign the fields of connected scripts graphically and test the implemented features. This will be addressed in more detail in section 1.2.

Following the installation of the above-mentioned Unity version, the editor must be set up according to the specifications of the developers of KSP in order to be able to export and embed the work. Therefore, the provided package `PartTools_PackageForModders`. `unitypackage` must be imported in the project [2, 3].

When installing the Unity Editor, the integrated development environment Microsoft Visual Studio 2019 Community can be downloaded as well. The .NET desktop development and game development with Unity workloads must be selected. Although Unity can handle scripts in .cs file format, KSP only loads dynamic-links libraries. For this, a new project template of type Class Library in .NET Framework 4.7.2 is opened in VS [4]. The structure of the project is based on the template from DMagic [5] and is displayed in figure 1, so that two projects are bundled into one assembly. Each project is compiled separately. The first class library is named `Joolyter.Unity` and contains the fields, properties and methods necessary for the user interface isolated from KSP. Loading the IDE in to and operation in the game is handled by `Joolyter.KSP`. Referenced dynamic-link libraries from KSP and Unity are located in `KSP_x64_Data\Managed\` of Kerbal Space Program's installation folder.

1.1 Design

The specifications state that the user interface must have a text editor and console output. This is ensured by choosing a layout that combines both and thus has parallels to popular integrated development environments, such as Visual Studio Code, PyCharm and MatLab. To meet the demand for a high level of accessibility for inexperienced users the complexity is kept as low as possible. Hence, the application will not have the same range of functions as the IDEs mentioned. Therefore a simplified layout without multi-layer menus can be used. Consequently, the main

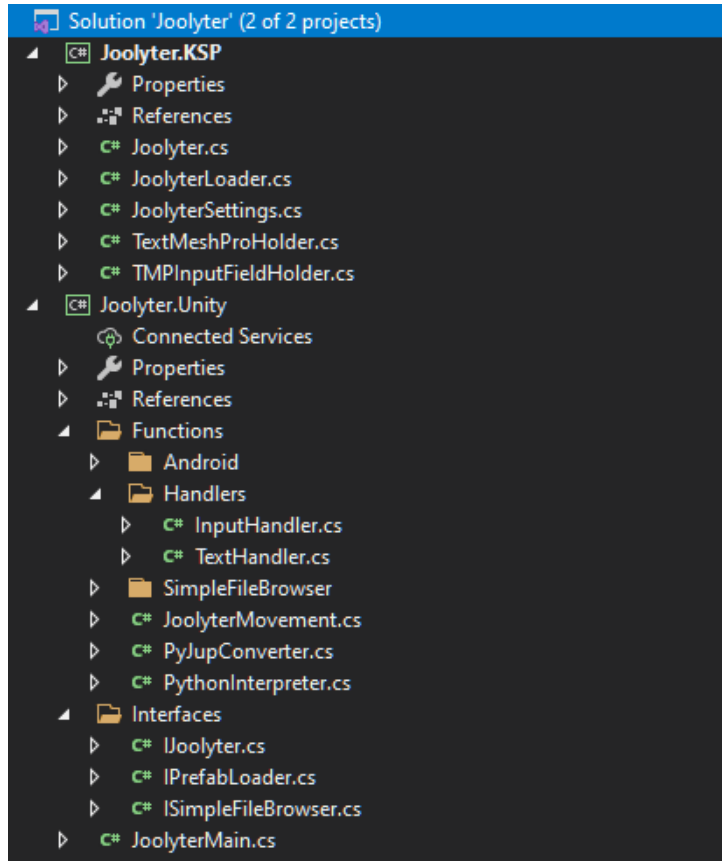


Fig. 1. File structure of the solution in Visual Studio

window has a title bar and menu bar at the top, below is the editor area. The console area is located at the bottom. A mock-up visualizes the layout in figure 2.

1.2 Functionality

All classes mentioned in 1.2 are part of in `Joolyter.Unity`. To use the editor with previously created files or to save edited files, functions for file management are necessary. As a result, a file menu consisting of four buttons is designed. These are titled New, Open, Save, and Save As. Pressing New resets the editor, Open allows the user to select and open an locally existing file. Save As works similarly, but the file path is defined. Clicking the Save button writes the contents of the editor to the opened file. In case Save is clicked but the file path is unknown it works as Save As. The input field's content are accessed in the data format `String`.

Using a file manager to determine the file path either at loading or saving complies with the demand for high accessibility. The open source project Unity Simple File Browser developed by Kula [6] meets the requirements, as the appearance differs only marginally from file managers common operating systems. The addon offers users to navigate through the systems data structure visually as well as create, rename, and delete files and folders. The file type filter above the Cancel button is set to default to Python files but can be changed to Jupyter Notebook files and no filter.

The standard IDE used in class is Jupyter Notebook. Even though a non-notebook layout is

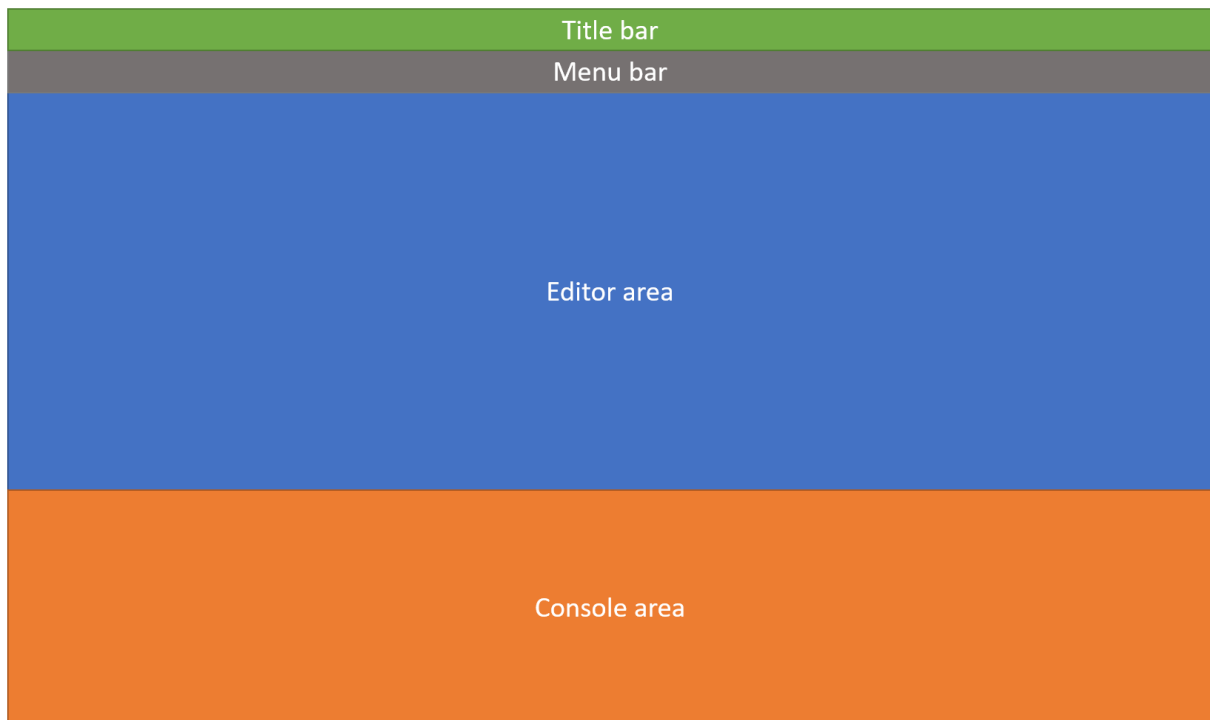


Fig. 2. Mock-up of UI

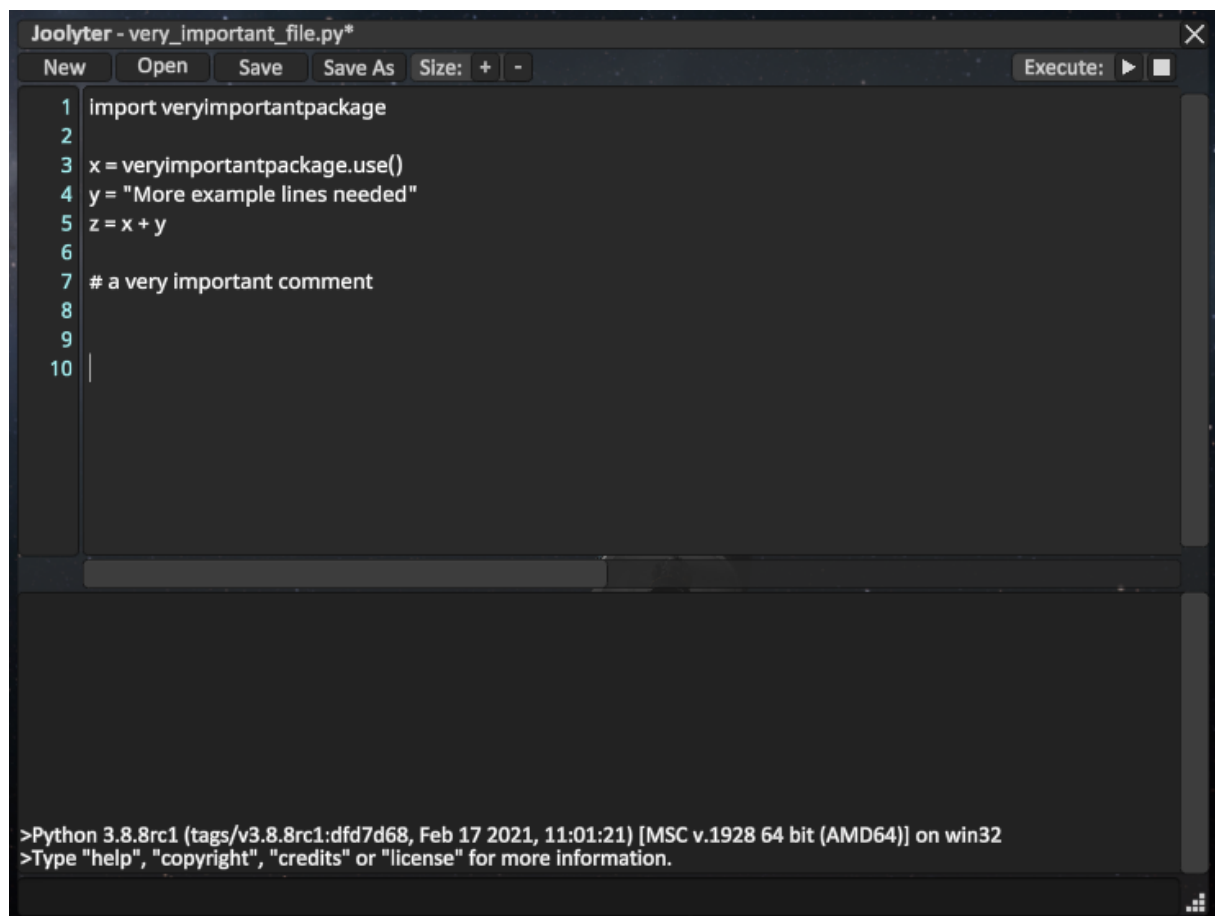
chosen here, users must be able to open `.ipynb`-files. To comply with this, a converter is used. Jupyter comes with the function `nbconvert` but it accepts Jupyter Notebook source files only. The package `p2j` represents an alternative because it supports conversions in both directions. The implementation in `C#` is done in the class `PyJupConverter`. It contains the method `Converter` that recognizes the file type and passes the appropriate arguments and parameters for calling `p2j` to the method `Execute` which creates and instantiates `System.Diagnostics.Process` with the given values. The class `Process` allows to interact with local and remote processes [7]. Joolyter always works with Python files. When loading a notebook file, a `.py` variant is created, which is processed and executed if necessary. As soon as the script gets saved as notebook, the Python file is written and then converted into an `.ipynb` file. The IDE saves the file type of loaded files, so clicking the Save button always saves in the opened file.

Figure 3 shows that loaded files are listed in the title bar by their name and changes to a saved file are indicated by a asterisk following the name.

Highlights in figure missing? (proficient imo)

Next to the file menu are two buttons to alter the font size of the editor and console area by manipulating the Text component's font size property.

The line numbering is also evident in figure 3. The feature supports users to debug scripts which represents another key requirement. It works by calling `JoolyterMain.LineNumbering()` on each change in the editor area's input field. The method counts the number of escape characters `\n` of type `char`. The type restriction prevents user induced input from being counted since only pressing the enter key produces `char` instances. Line numbers are printed in the field on the left



The image shows a screenshot of the Joolyter Python editor. The window title is "Joolyter - very_important_file.py". The menu bar includes "New", "Open", "Save", "Save As", "Size: + -", and "Execute: ▶ ■". The code editor displays the following Python code:

```
1 import veryimportantpackage
2
3 x = veryimportantpackage.use()
4 y = "More example lines needed"
5 z = x + y
6
7 # a very important comment
8
9
10 |
```

Below the code editor is a terminal window showing the Python version and environment information:

```
>Python 3.8.8rc1 (tags/v3.8.8rc1:dfd7d68, Feb 17 2021, 11:01:21) [MSC v.1928 64 bit (AMD64)] on win32
>Type "help", "copyright", "credits" or "license" for more information.
```

Fig. 3. Joolyter Editor in use

accordingly.

No explanation of mode of action needed? (scientific approach/replicability)

The execution menu in the upper right corner and the Python console are closely related. The mode of action of the console is described first.

The foundation is `PythonInterpreter.StartPython()`, which is called in `JoolyterMain.Start()`. In it, a Python process is started and all errors and outputs are stored in a public list. The list is checked for content in every frame in `JoolyterMain` and if necessary the elements of the list are given to `WriteLineInConsole()` of the same class consecutively. The list operates as a courier because each entry is deleted after it was passed. `WriteLineInConsole()` adds a new text component, that is stored in the prefab `OutputLineText`, to the scrollview of the console area, in which the passed text is subsequently written. This sequence is repeated until the list is empty [8].

Too much detail? (important for replicability imo)

To interact with the Python console, the input field at the bottom of the window is used. Pressing `Return` calls `JoolyterMain.SubmitInputLine()` when the field was selected. The method passes the content to `WriteLineInConsole()` which prints it in the console. Afterwards the submitted string is compared to the term `clear`, if true all contents of the console and assigned variables in Python are deleted. Other inputs are written to the Python process asynchronously in the `PythonInterpreter` class and get printed to the console.

Too much detail? (important for replicability imo)

Running a script from the editor by clicking the Play button also uses the `StartPython()` method. However, unlike at the call to run the console, a file path to the script is passed here. Within `StartPython()` this is then added as argument to the process which leads to the execution of the script in Python. Errors and output are passed to the console in the same way as before. After running the code the console can be used to interact with variables assigned inside the script as its process is running. Executing a file automatically resets the Python process. If an infinite loop occurs or the execution of a script or console command is to be terminated prematurely, the Stop button ends and restarts the process.

In combination the editor, Python interpreter, and console offer users to create, execute, debug, as well as save Python and Jupyter Notebook scripts, which students are familiar with.

The requirement for very high levels of usability and portability, in the context of a deployment of the software on personal devices - mostly small screen laptops - and the university workstations, must be fulfilled. This can be supported by making the window adjustable in size and position. The class `JoolyterMovement` adds the functionality and is based on Unity Simple File Browser. When the window is opened for the first time and after changing its position or dimensions, `JoolyterMain.EnsureWindowIsWithinBounds()` is called, which checks that the window is completely visible and does not fall below the minimum size [9].

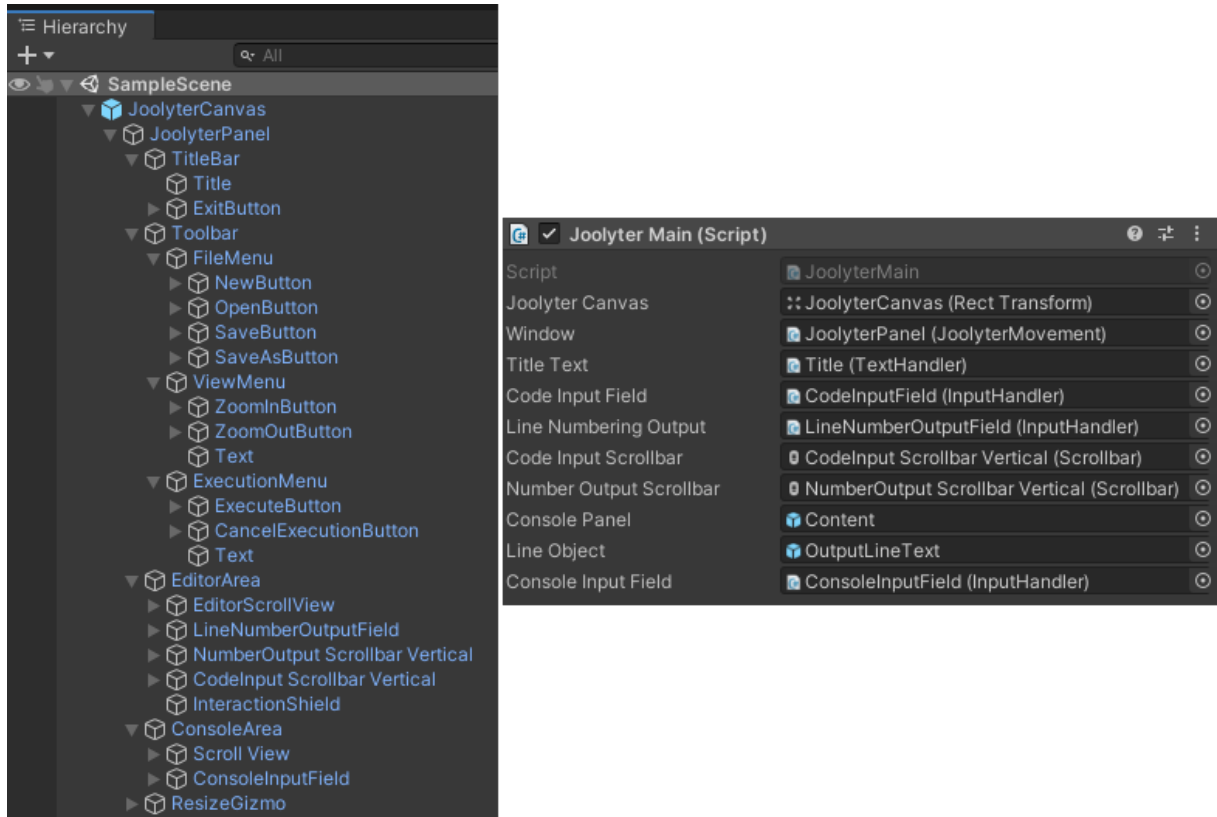


Fig. 4. Unity hierarchy and assignment of JoolyterMain component

The design choices made in 1.1 and added functionality lead to the hierarchy of Unity game objects shown on the left of figure 4. As already mentioned in the beginning of section 1 fields can be assigned in Unity dragging a game object from the hierarchy to a field of a script. This is displayed on right side of figure 4 using the example of **JoolyterMain**. Underneath the word "Script" in grey are the names of serialized and public fields in **JoolyterMain**. For increased readability Unity changes the names that are displayed thus, `_joolyterCanvas` becomes "Joolyter Canvas". In the dark box next to the field's name is the game object from the hierarchy. Behind that in brackets is the type of the field. The procedure ensures that not all game objects and their components need to be searched when loading the prefab in Kerbal Space Program. This reduces the susceptibility to errors and improves runtime performance. To be able to set the variables, the dynamic-link library of **Joolyter.Unity** must be built and imported into the Unity project.

Interfaces are used to communicate between **Joolyter.Unity** and **Joolyter.KSP**. This allows to call methods and properties in **Joolyter.Unity** that are implemented in **Joolyter.KSP** without causing circular references.

Tests in Unity's Game View show the basic functionalities of the IDE. To make the editor area scrollable, Text Mesh Pro's InputFields are used. TMP is a Unity asset that replaces Unity's text components. Originally, it could be purchased from the Unity Asset Store, as it was not developed by Unity Technology. In 2017, the asset was acquired by and integrated into Unity [10]. However, there are some compatibility issues between the paid and free variants of Text Mesh Pro. As KSP uses the paid version TMP cannot be used in the Unity Assembly, as

mentioned in sub-subsection 1.1. The user DMagic of the Kerbal Space Program forum has developed a way to use the TMP elements anyway. To do this, a script is added to all game objects with `Unity.UI.Text` components, which both tags them and contains the methods, as part of the `Joolyter.Unity.TextHandler` class, to dynamically adjust the content using generic `UnityEvents` [11]. After the loading of the user interface in KSP has been explained in the next sub-subsection, the mode of operation will be discussed in more detail.

In the final step in Unity, the prefabs are combined to `AssetBundles` that can be loaded in to KSP. For the core application, `joolyterassetbundle` is created. It contains the prefabs `JoolyterCanvas` and `OutputLineText`. A second `AssetBundle` called `simplefilebrowser` includes all prefabs of Unity Simple File Browser. The `AssetBundles` are created with the Asset Compiler, which is part of the `PackageForModders` installed at the beginning [5].

Too much detail? (important for replicability imo)

1.3 Loading addon in to KSP

After creating `Joolyter.Unity` and the `AssetBundles`, the class library `Joolyter`. KSP is developed. It includes the classes `TextMeshProHolder`, `TMPInputFieldHolder`, `JoolyterLoader`, `Joolyter` and `JoolyterSettings`. The declarative tags in the non-holder scripts, instruct Kerbal Space Program when to load the script.

`JoolyterLoader` implements the interface `Joolyter.Unity.IPrefabLoader` and loads the game objects for rendering the main window from the `AssetBundle` `joolyterassetbundle`. In addition, all `TextHandler` and `InputHandler` components are processed in the loaded prefabs by replacing the `Text` and `InputField` components with the Text Mesh Pro counterpart. For this purpose the extension classes `TextMeshProHolder` and `TMPInputFieldHolder` are used respectively. Together with the handler classes, these provide the possibility to update the new TMP elements via generic Unity events from classes in `Joolyter.Unity` [5, 11].

In order to open the addon's window in Kerbal Space Program, a button as well as methods to open and close the window must be added to the internal toolbar, to which the corresponding methods are attached. This is the main purpose of the class `Joolyter`. The method `OnGUIAppLauncherReady()` provides the button at in flight scenes with a specified icon [12].

Since errors occurred repeatedly in tests, the prefabs of `joolyterassetbundle` are processed again in `PostProcessPrefab()`. Thereby the listeners and events of the input fields are bound, some properties of the `TMP_Text` and `TMP_InputField` components as well as the scrollbars of the editor area are assigned. Due to its error-proneness, the method is intentionally not optimized in terms of runtime to ensure maximum readability and thus keeps debugging simple.

The folder and file structure shown in Figure 5 ensures the loading of the texture as well as `AssetBundles` in `Resources` and the dynamic-link libraries in `Plugin` if places in KSP's installation folder in `GameData`. `backup` catches unsaved files when the game exits under the condition that `Joolyter.KSP.Joolyter`. `OnDestroy()` is called. This is not the case when the game is closed via direct task termination.

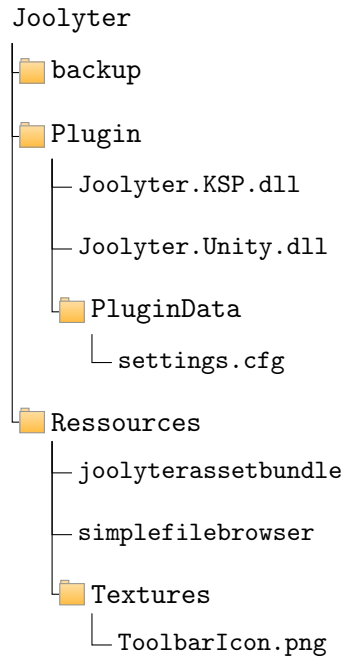


Fig. 5. File structure of Joolyter in GameData

2 Learning Application

This section covers the workflow for creating tasks with an example and explains how to interact with Kerbal Space Program using Python.

2.1 Mission design

Kerbal Space Program offers the possibility to design and share missions in combination with the paid extension Making History. The Mission Builder allows the selection of spacecrafts, setting start conditions, failures, milestones, goals messages to the player and other events as well as connecting them with logical operators. Therefore it relies on flow- based programming. The developers have published several tutorials on different media, so there are guided in-game tutorials under *Start Game -> Play Missions -> Stock Missions -> Tutorials* as well as videos and texts in which the process is explained by means of an example [13, 14]. For demonstration purposes, two missions, based on homework from the Fundamentals of Astronautics class in the winter semester 2020/2021, were created as part of this thesis.

The first mission requires the user to calculate the change of velocity required for a Hohmann transfer. The setting of the mission is loosely based on the Hubble Space Telescope, but follows the humorous approach of the game. On a circular parking orbit with an altitude of 150 km AMSL, an engine of the launch vehicle, carrying the telescope, fails, so that it has to reach the target orbit under its own power. The user must to calculate the necessary change of velocity in both maneuvers of a Hohmann transfer using a Python script. The secondary goal of the first mission is to familiarize the user with the operation of the learning application.

The second mission is more complex. The change of velocity for a transfer to a stationary orbit is to be calculated for a satellite, which is on an inclined low circular orbit. This requires two maneuvers to change the semi- major axis as well as the inclination. To determine the

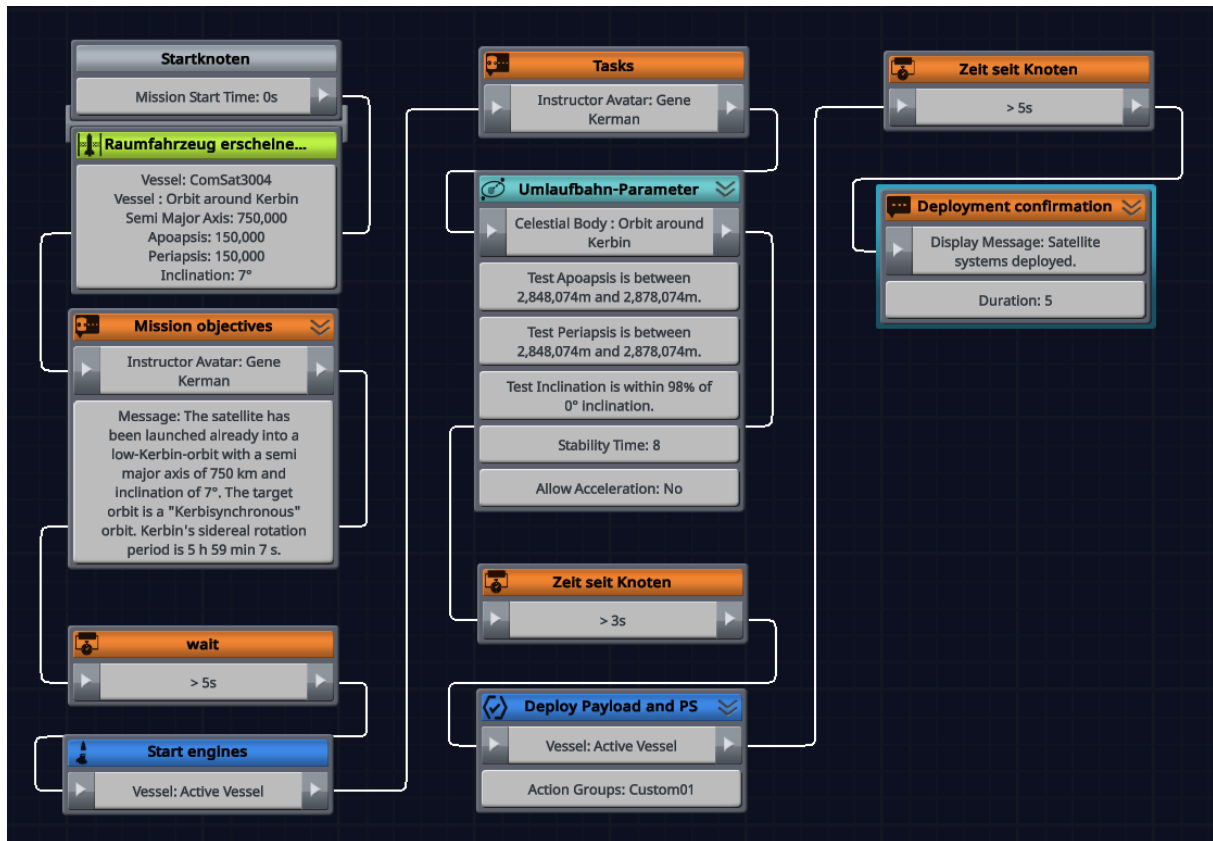


Fig. 6. Mission Two - Inclination Change in the KSP Mission Builder

parameters of the target orbit, the duration of a sidereal day is given. The most energetically efficient transfer variant is to be chosen from three given options. Thereby the pure inclination change is compared with the combined maneuvers. In the process the correlation of orbit altitude, velocity and the effect on the velocity requirement for inclination adjustments is deepened. This mission in the Mission Builder is shown in Figure 6. In the first node the start time, as well as the spacecraft and its orbit parameters are defined. At start a dialog message is shown informing the player about the mission objective. Five seconds after displaying the message, the vehicle's engines are activated and another dialog message is opened specifying the task for the player, representing the actual start of the mission from the player's perspective. The following node "Orbit-Parameter" checks if the specified values are reached. This corresponds to a stationary orbit with respect to the planet Kerbin with a maximum orbit altitude deviation of 15 km. It must be kept stable for eight seconds, during which time no acceleration may occur. As long as the target orbit is not reached, no continuation is made. After waiting for three more seconds, the solar and communication modules are deployed. This node's effect is purely cosmetic. A text message confirms the successful deployment of the payload marks the end node and triggers a window congratulating the player on completing the mission.

All missions are saved to the Kerbal Space Program folder in `Kerbal_Space_Program\Missions\` by default. In addition they can also be exported from the Mission Builder, which ensures no progress saved in `persistent.mission` [15]. The directory of a mission can be seen in figure 7 using the example of Mission Two. To import spacecrafts from other saves their `.craft` file needs to be in placed in the respective folder. **VAB** is short for vehicle assembly building and **VAB** stands

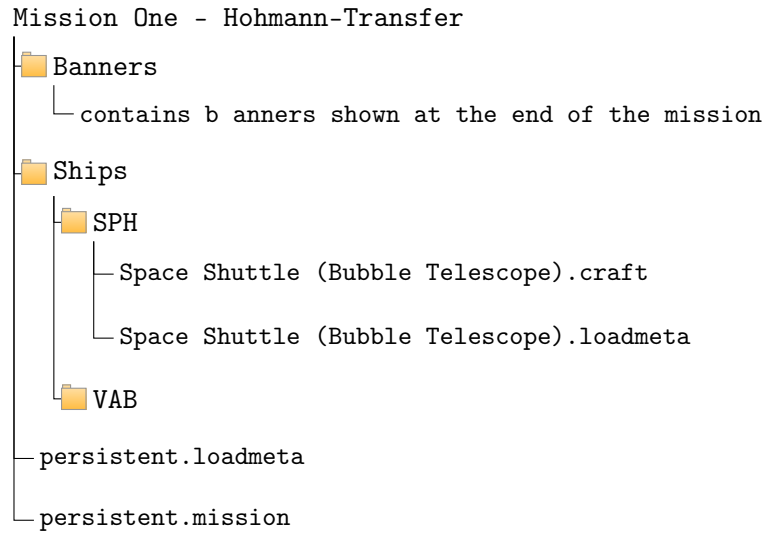


Fig. 7. File structure of Mission One - Hohmann-Transfer

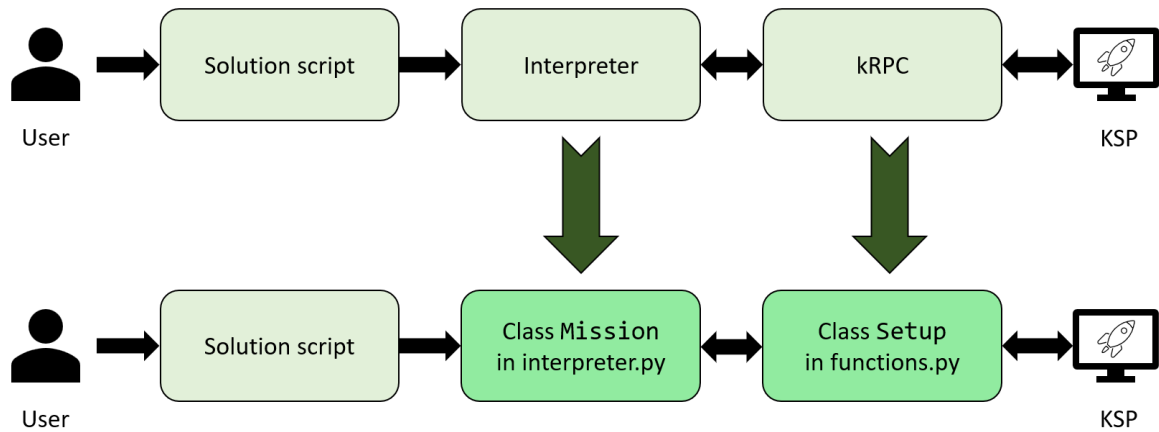


Fig. 8. Solution code processing schematics

for spacecraft hangar. Beyond the on-screen instructions, both missions have tasks in written form in the appendix to allow testing in a class environment.

2.2 Parser - kRPC

In order to enable the user to control KSP with numerical values, they must be interpreted logically. This allows KSPs to be used via the API to pass control commands. Kerbal Space Program has a built-in application programming interface (API) which supports C# natively [16]. Since the specification dictates that the application can be extended with the use of Python, the API must be made accessible without the use of other languages. kRPC - Remote Procedure Call Server for KSP (kRPC) is a third party application that covers exactly this use case [17]. This results in the following structure in the application and operation of the learning program: The user develops a solution script for a specific task. A parser interprets the results and thus, commands can be given to KSP using kRPC. This is illustrated in the top row of figure 8. A disadvantage of kRPC is that compatibility issues occur in late Python environments. There

are reports that kRPC runs on `Python > 3.8`, but in the course of this work it was not possible to connect to KSP when using versions 3.9 or 3.10. Even with `Python 3.8`, installation and operation are not possible without further adjustments of dependencies. By specifying the version of dependencies when building the Python package that contains the parser and implements kRPC, full functionality can be ensured without causing extra work for users. The development of the package is discussed at the end of this sub-subsection.

The green arrows in figure 8 visualize that the parser and the integration of kRPC need to be developed. The class `Setup` in `functions.py` implements kRPC and contains the functions and properties to communicate with KSP. Since the parsers for the results are unique for each scenario, `mission.py` file holds a function for each mission which is called in the student's script to submit the results to KSP. All functions use an instance of the `Setup` class to communicate with the game.

When `Setup` gets instantiated, a server connection to KSP is established and streams to access telemetry data in real time are set up. The streams are private and the current values can be accessed through properties. This is done by misusing the `@property` decorator, as it is not the stream itself that is returned, but its current value. Furthermore, the class acts as a collection for reusable functions for the parser. Exemplary for this is `impulse_maneuver(t_maneuver : int, dv_pro = 0.0, dv_normal = 0.0, dv_rad = 0.0, lead_time = 30) -> bool`. After passing the mathematical moment of the maneuver `t_maneuver` and the change of velocity in its spatial components in the orbital reference system, a maneuver node is created in KSP, the spacecraft is aligned and the burn is performed. `lead_time` represents the duration a automatic time warp is ended before the burn starts. Automatic time warp is deactivated on source code level, but the functionality is provided. The return value indicates whether the maneuver could be performed successfully.

The mode of action of parsing functions is explained using the second mission as an example again. The most energetically sensible transfer to the target orbit is to be determined. This may require different maneuvers in a certain order and thus, students must implement instances of the heirs of the class `ImpulseManeuver`. Available for this purpose are `VelocityChangeManeuver`, `InclinationChangeManeuver`, and `CombinedManeuver`, each of which represents the name-giving maneuver. The classes are instantiated with the values of velocity and change of velocity needed to calculate the directional change of velocity in the orbital reference frame. The determined instances are passed to the following function which represents the main parser: `mission_two(maneuver1 : ImpulseManeuver, maneuver2 : ImpulseManeuver, maneuver3 : ImpulseManeuver = None,) -> bool`. The return value provides information on the success of the transfer after all maneuvers were carried out.

The class `Setup` is instantiated early in `mission_two()`. The given instances of the maneuver classes are processed one after the other. Thereby it is checked if the ascending and descending nodes coincide the apsides using properties of the property `Setup.vessel`, the active spacecraft in KSP, an instance of `krpc.types.Vessel`. This demonstrates that the use of kRPC is possible outside the `Setup` class. If the request returns `True` the maneuver is executed at the next node. The directional change of velocity needed to initiate the maneuver with `Setup.impulse_maneuver()` is provided by `ImpulseManeuver.direct_dv()`. Thus, the class

`ImpulseManeuver` is executing a part of the parsing. After processing all given maneuvers, a control loop evaluates the success of the transfer.

To follow the requirement specification, the Python scripts are bundled in a package to provide a high degree of portability and a user-friendly way of installation of the specific dependencies of `kRPC`, `kRPC` itself and the package `p2j` which is used in the user interface's background. This is done by setting these packages as requirements for `JoolyterDemo`. The package itself contains the modules `functions.py`, `maneuver.py`, `missions.py`, `mission_solution.py` and `__init__.py` which defines that the usable maneuver classes as well as the functions `mission_one()` and `mission_two()` are automatically imported when importing the package. The wheel module is used to create the package as a wheel file, which can be installed by the students using `pip`. Installing additional packages in this way is an important skill and thus does not conflict with the requirement to provide maximum accessibility.

The module `mission_solution.py` contains sample solutions for both demonstration missions to allow alpha testers without knowledge of orbital mechanics.

3 Installer development

In order to install Kerbal Space Program as educational software, Joolyter's [Figure 5] as well as `kRPC`'s folders must be placed in `GameData` and folders of the two demonstration missions [Figure 7] must be placed in `Missions` in KSP's installation folder. To ensure a simple setup for students without experience in Python a graphical installer of the adequate version 3.8 is included. Furthermore, the functions that students use to start visualizing the calculated values in KSP are part of the JoolyterDemo package, whose wheel file is also required. This way an installation using `pip` is possible. To support users, a manual has been created that contains step-by-step instructions and explains how to use KSP, the Joolyter IDE, and the JoolyterDemo package. This and the HTML documentation of JoolyterDemo are in the `ReadMe` folder. Additionally, the installation folder contains the tasks in written form with `MissionBriefing.pdf`. To avoid users having to browse unknown folders to open Kerbal Space Program and the documentation of JoolyterDemo, batch files are used, because Microsoft Windows shortcuts only support absolute file paths. `KSP_Launcher.bat` starts Kerbal Space Program and `JoolyterDemoDocumentation.bat` opens the HTML documentation of JoolyterDemo. To be able to distribute the above files to a large number of users, they are grouped together in the `InstallationContainer` folder, as shown in Figure 1. Compressing the folder in ZIP file format reduces its size to 2.57 GB including the game Kerbal Space Program. This allows hosting on the university's own cloud platform TUBcloud and the download from the TU Berlin's WiFi network takes only a few minutes.

4 Extensibility

Operators of the learning application must be able to add new tasks according to the specifications. This should be made as simple as possible. Although it could be avoided that people who want to extend the software with new missions have to learn `kRPC`, it would be necessary to make all functionalities of `kRPC` accessible with new syntax. Therefore, the workload for teachers would merely shift, due to the complexity remaining the same. In addition, the amount

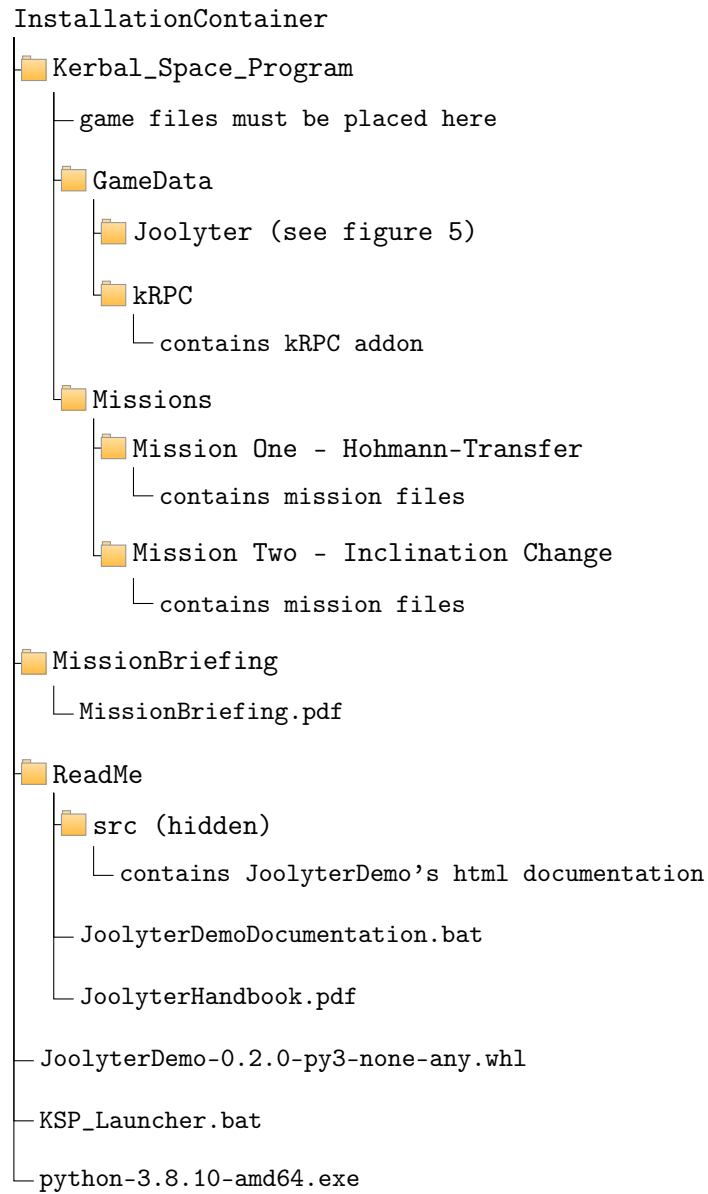


Fig. 9. File structure of InstallationContainer for Students

of work would be immense. Based on the above arguments, as well as the detailed documentation with instructions and an active online community, the effort is considered acceptable [17, 18]. In addition, html documentations were created for both the C# code of the IDE and the Python-based processing of the tasks. The existing documentation and instructions in the area of UI development are supplemented by the documentation of the Joolyter IDE to ensure a high level of accessibility for developers without knowledge of C#. Thus, the IDE can be serviced by previously untrained personnel as well as optimized and extended beyond. In the area of result interpretation and communication with Python, the documentation in the current extent ensures the comprehensibility of the code. In the case of the development of further tasks and a growing scope, the documentation can simplify the creation of new tasks thus, a script that automates the creation of the documentation using sphinx is included.

References

- [1] Intercept Games. “Kerbal Space Program 1.12.4 Releasing 11-2-2022,” Take-Two Interactive Software, Inc. (Nov. 1, 2022), [Online]. Available: <https://forum.kerbalspaceprogram.com/index.php?/topic/210392-kerbal-space-program-1124-releasing-11-2-2022/#comment-4193952> (visited on Nov. 5, 2022).
- [2] JPLRepo. “Official PartTools,” Take-Two Interactive Software, Inc. (May 11, 2017), [Online]. Available: <https://forum.kerbalspaceprogram.com/index.php?/topic/160487-official-parttools/#comment-3052773> (visited on May 20, 2022).
- [3] Unity Technologies. “Unity - Download Archive.” (2022), [Online]. Available: <https://unity3d.com/get-unity/download/archive> (visited on Sep. 16, 2022).
- [4] Linx. “How I Make My Mods / Plugins for Kerbal Space Program.” (May 5, 2020), [Online]. Available: <https://youtu.be/i0I7MhOM7mg?t=9> (visited on May 19, 2022).
- [5] DMagic. “Unity UI Creation Tutorial,” Take-Two Interactive Software, Inc. (Nov. 2, 2016), [Online]. Available: <https://forum.kerbalspaceprogram.com/index.php?/topic/151354-unity-ui-creation-tutorial/#comment-2835301> (visited on Jul. 18, 2022).
- [6] S. Y. Kula. “Unity Simple File Browser,” GitHub, Inc. (n.d.), [Online]. Available: <https://github.com/yasirkula/UnitySimpleFileBrowser> (visited on Jun. 14, 2022).
- [7] “Process Class (System.Diagnostics) | Microsoft LearnMicrosoft Corporation,” Microsoft Corporation. (n.d.), [Online]. Available: <https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.process?view=netframework-4.7.2> (visited on Sep. 22, 2022).
- [8] Soupertrooper. “Unity 3d Tutorial: Chat Box and Message System,” Google LLC. (Jan. 8, 2017), [Online]. Available: <https://youtu.be/IRAeJgGkjHk> (visited on May 25, 2022).
- [9] S. Y. Kula. “Unity Simple File Browser - FileBrowserMovement.cs,” GitHub, Inc. (Oct. 26, 2020), [Online]. Available: <https://github.com/yasirkula/UnitySimpleFileBrowser/blob/master/Plugins/SimpleFileBrowser/Scripts/FileBrowserMovement.cs> (visited on Jun. 14, 2022).
- [10] Unity for Games [@unitygames]. “We’re pleased to announce that popular @unity- asset-store tool TextMesh Pro is now free! [...],” Twitter. (Feb. 28, 2017), [Online]. Available: <https://twitter.com/unitygames/status/836625140054179842> (visited on Sep. 16, 2022).
- [11] DMagic. “Unity UI Creation Tutorial,” Take-Two Interactive Software, Inc. (Dec. 15, 2016), [Online]. Available: <https://forum.kerbalspaceprogram.com/index.php?/topic/151354-unity-ui-creation-tutorial/#comment-2888882> (visited on Jul. 18, 2022).
- [12] “Kerbal Space Program: KSP.UI.Screens.ApplicationLauncher Class Reference,” Take-Two Interactive Software, Inc. (Dec. 10, 2021), [Online]. Available: https://www.kerbalspaceprogram.com/ksp/api/class_k_s_p_1_1_u_i_1_1_screens_1_1_application_launcher.html#a5a512ce46a2e732017cc6c3faf26754 (visited on Sep. 17, 2022).

- [13] Kerbal Space Program. “KSP Making History Expansion - How to Create with the Mission Builder,” Google LLC. (Mar. 8, 2018), [Online]. Available: <https://youtu.be/xyR0iNr-Uug> (visited on Sep. 17, 2022).
- [14] kerbaldevteam. “Kerbal Space Program: Making History Expansion - How to create in the Mission Builder,” Tumblr, Inc. (Feb. 21, 2018), [Online]. Available: <https://kerbaldevteam.tumblr.com/post/171138611984/kerbal-space-program-making-history-expansion> (visited on Sep. 17, 2022).
- [15] Kerbal Space Program. “KSP Making History Expansion - How to Play and Share with the Mission Builder,” Google LLC. (Mar. 8, 2018), [Online]. Available: <https://youtu.be/5oKtOHLkRMM> (visited on Sep. 17, 2022).
- [16] “Kerbal Space Program: KSP.UI.Screens.ApplicationLauncher Class Reference,” Take-Two Interactive Software, Inc. (Dec. 10, 2021), [Online]. Available: <https://www.kerbalspaceprogram.com/ksp/api/> (visited on Sep. 22, 2022).
- [17] djungelorm. “kRPC Documentation - kRPC 0.4.8 documentation,” GitHub, Inc. (2017), [Online]. Available: <https://krpc.github.io/krpc/> (visited on Aug. 29, 2022).
- [18] djungelorm. “[1.5.x to 1.2.2] kRPC: Control the game using C#, C++, Java, Lua, Python, Ruby, Haskell, C (Arduino)... (v0.4.8, 28th October 2018),” Take-Two Interactive Software, Inc. (Jan. 26, 2016), [Online]. Available: <https://forum.kerbalspaceprogram.com/index.php?/topic/130742-15x-to-122-krpc-control-the-game-using-c-c-java-lua-python-ruby-haskell-c-arduino-v048-28th-october-2018/> (visited on Aug. 26, 2022).